

El desarrollo de aplicaciones software basado en líneas de productos (SPLs) permite a los desarrolladores modelar sistemas en función de la relación entre sus componentes. Así, es posible estudiar la variabilidad y características comunes entre ellos. Este paradigma de desarrollo se remonta a finales de los años 80. Desde entonces, se han realizado numerosos aportes sobre las distintas maneras de modelar las relaciones entre sus componentes. Inicialmente, estas relaciones fueron definidas de forma gráfica, sirviendo como soporte para la toma de decisiones en las fases iniciales del diseño de los productos. Sin embargo, para detectar errores de diseño en fases tempranas del proceso de desarrollo, es necesario realizar comprobaciones automáticas sobre estos modelos.

El objetivo principal de esta tesis doctoral es definir nuevos métodos formales para la representación de SPLs, teniendo en cuenta aquellos modelos que carezcan de formalismos matemáticos para su representación.



UNIVERSIDAD
COMPLUTENSE
MADRID

Facultad de Informática

Modelando la variabilidad: Métodos formales para la representación de líneas de productos software

Tesis doctoral

Autor:

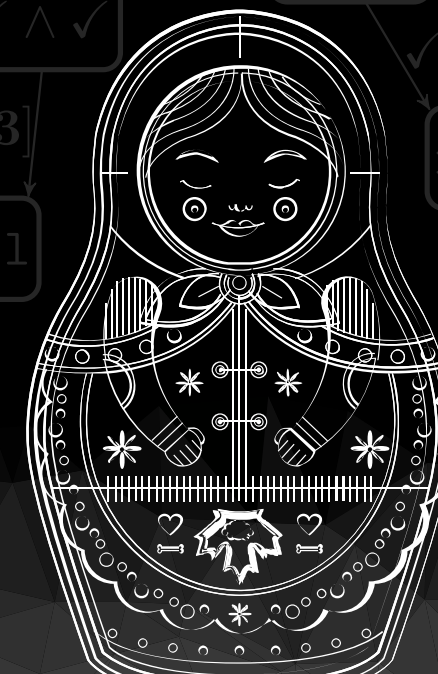
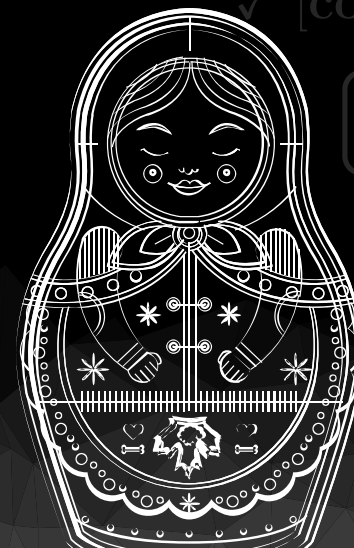
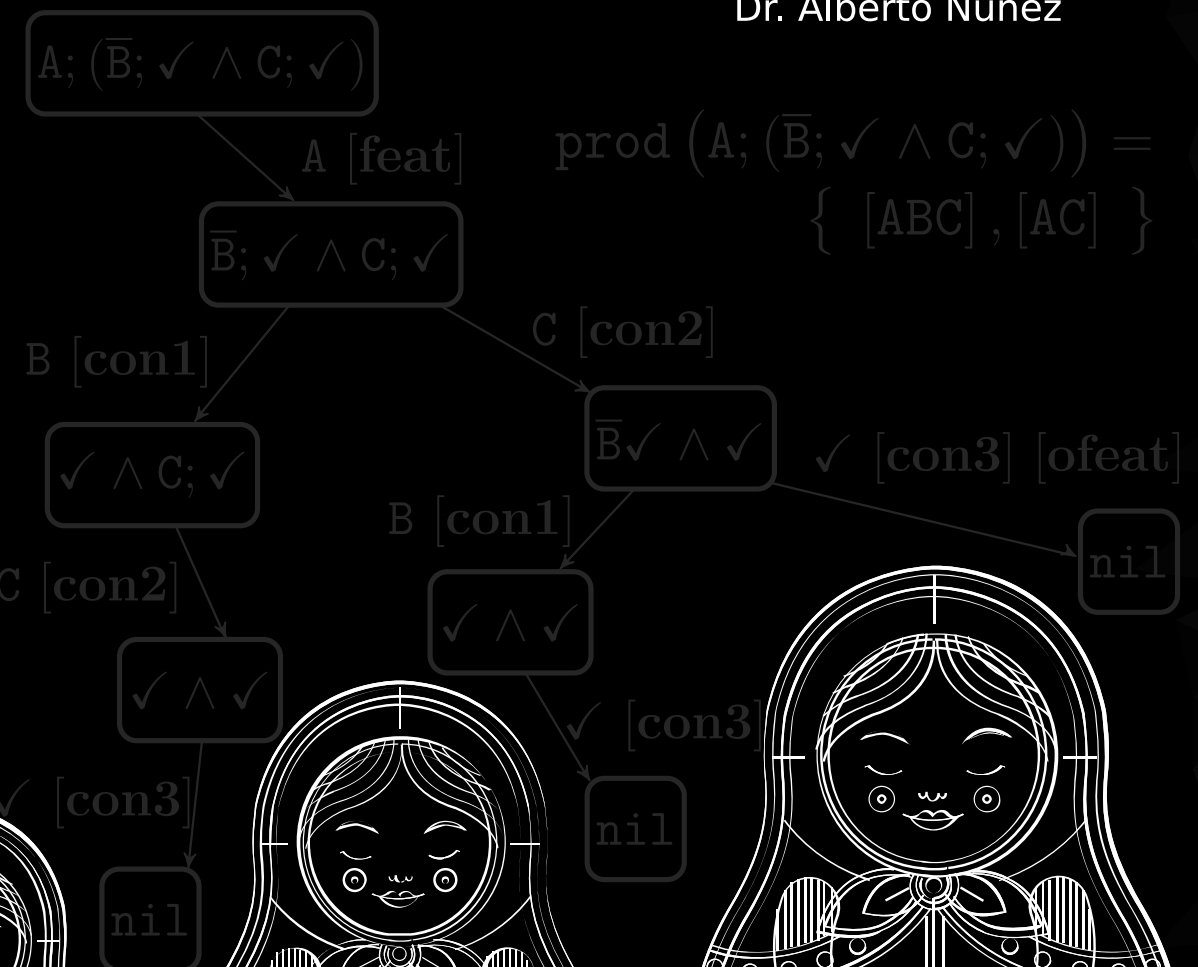
Carlos D. Camacho González

Directores:

Dr. Luis Llana

Dr. Alberto Núñez

$$P = A; (\bar{B}; \checkmark \wedge C; \checkmark)$$



Modelando la variabilidad: Métodos formales para la representación de líneas de productos software - 2017

$$A; (\bar{B}; \checkmark \wedge C; \checkmark) =_E [\text{CON2}], [\text{CON1}] \quad A; C; (\checkmark \wedge \bar{B}; \checkmark) =_E [\text{CON2}], [\text{CON5}]$$

$$A; C; \bar{B}; \checkmark =_E [\text{PRE2}]$$

$$A; C; (B; \checkmark \vee \checkmark) =_E [\text{PRE3}]$$

$$A; (C; B; \checkmark \vee C; \checkmark) =_E [\text{PRE1}]$$

$$A; (B; C; \checkmark \vee C; \checkmark)$$

$$[\bar{B}; \checkmark \wedge C; \checkmark] = [\cdot \wedge \cdot] (\{\emptyset, \{B\}\}, \{\{C\}\}) = \{\{C\}, \{B, C\}\}$$

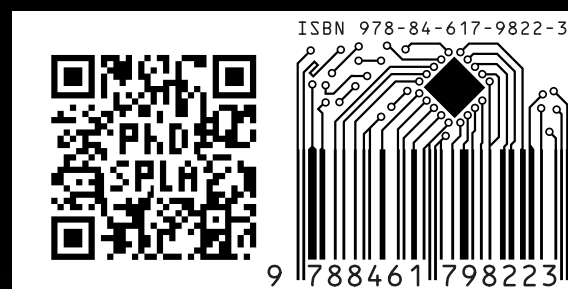
$$[A; (\bar{B}; \checkmark \wedge C; \checkmark)] = [A; \cdot] ([\bar{B}; \checkmark \wedge C; \checkmark]) =$$

$$[A; \cdot] (\{\{C\}, \{B, C\}\}) =$$

$$\{\{A, C\}, \{A, B, C\}\}$$



UNIVERSIDAD
COMPLUTENSE
MADRID



ISBN 978-84-617-9822-3

9 788461 798223

Modelando la variabilidad

Métodos formales para la representación de
líneas de productos software



UNIVERSIDAD
COMPLUTENSE
MADRID

Memoria que presenta para optar al título de Doctor en Ingeniería Informática

Carlos Delfín Camacho González

Dirigida por los doctores

Luis Llana Díaz y Alberto Núñez Covarrubias

Universidad Complutense de Madrid
Facultad de Informática
Departamento de Sistemas Informáticos y Computación
Madrid, 2017

Esta página ha sido intencionalmente dejada en blanco.

*A aquellos que más quiero,
Cristina, mis padres y hermanas.*

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

El desarrollo de esta tesis doctoral ha sido posible gracias al apoyo de muchas personas y, por esa razón, me gustaría expresarles mi absoluto y sincero agradecimiento.

Estos últimos 5 años han sido una experiencia ardua, gratificante y sobre todo de aprendizaje, que me ha permitido crecer tanto profesional como personalmente. Hoy puedo ver que todo es posible gracias al trabajo en equipo. Sin él, estos agradecimientos no existirían. No hay manera en que pueda plasmar en papel, lo agradecido que estoy con todas las personas que de una u otra manera han hecho esto posible, sin embargo, al menos, he de nombrarlos.

Ante todo, agradecer a los profesores Luis Llana, Alberto Núñez y César Andrés por toda su ayuda, apoyo y colaboración, sin ellos, la culminación de este trabajo de investigación habría sido imposible.

A mis padres, quiero agradecerles todo lo que me han dado, ya que ellos han hecho posible todo esto. También a mi familia que sé que siempre ha estado a mi lado para apoyarme y ayudarme en lo que necesite.

A Cristina, quien representa el pilar y el origen de mi felicidad, su cariño y apoyo incondicional fueron indispensables para terminar este proyecto.

También, me gustaría hacer un reconocimiento especial a todas las personas que me han acompañado a lo largo de mis andanzas fuera de Venezuela. Mary, Rubén, Jessica, Harold, Rubencito, Mario, Carlos, Morla, Alvaro, Angel, Ka, Irina, Josito, Marta, Paula, Cristóbal, Agustín, Carolina, Francho y muchos más.

Para finalizar, quiero igualmente agradecer a las empresas que en el transcurso del desarrollo de este proyecto me han ayudado de manera directa o indirecta, ellas son, Red Hat, Frontiers y Santa Mónica Media.

A todos, muchas, muchísimas gracias.

Esta página ha sido intencionalmente dejada en blanco.

Abstract

- Title: Variability modeling: Formal methods to represent Software Product Lines.
- Author: Carlos D. Camacho González
- Supervisors: Dr. Luis Llana Díaz and Dr. Alberto Núñez Covarrubias.
- Institution: Complutense University of Madrid, Spain.
- Center: Faculty of Informatics.
- Program: Ph.D. in Informatics Engineering.

Research justification. The software development paradigm focused on product lines (SPLs) allows developers to model systems based on product components and to study their commonality and variability. This paradigm dates to late 80's and, since then, several contributions have been made for modeling relationships between components [7]. Initially, these relationships were defined in a graphical way, serving as a basis for decision making purposes in several design phases of the development cycle. In order to detect design errors in early phases of the development cycle, it is necessary to use formal methods allowing automated analysis over the SPLs. Thus, in this thesis we propose to define formal methods for modeling SPLs, which allows to automatize its processing and analysis.

Questions that motivate the research. From the conditions described above, the following questions arise to motivate the development of this Ph.D thesis. Is it possible to define a formal framework to model FODA and extend it to model other variability models?, How to analyze a SPL without generating all its valid products?, What is the effect of pre-processing the algebra terms before obtaining the valid products?, What are the advantages of comparing products based on their cost?, Is it possible to determine the probability of occurrence of a characteristic in any model?. Once described these questions we proceed to define the global and specific objectives of this doctoral thesis.

Objectives. The main objective of this doctoral thesis¹ is to define new formal methods to represent **SPLs**, taking into account models lacking mathematical formalism for their representation. Based on this general objective, the following specific objectives are defined: i) defining a formal representation for **FODA** that allows to perform automated analysis on its structure; ii) modeling the production cost within the building process of the valid products; iii) representing probabilities within the algebra terms; and iv) showing different implementations of the described formalism depending on the information that is necessary to generate.

Contributions. This doctoral thesis has allowed to define a theoretical framework to formally represent and describe **FODA** diagrams, including its syntax, the translation process between **FODA** and **SPLA**, its set of semantic rules, implementation and possible practical case studies. Thus, given the need to represent costs in the models, the **SPLA** definition was extended to support costs. Similarly, the equivalence between these two models, their implementation and possible practical cases are also shown. In order to finalize the contributions of this Ph.D. thesis, a new extension was proposed to identify those components and valid configurations being more or less frequent in the product line.

Keywords: Software Product Lines, FODA, Formal methods, Process algebras, Operational semantics, Denotational semantics, Axiomatic semantics.

¹The content of this Ph.D. dissertation and all its supports are available online at the following URL: <http://ccamacho.github.io/phd/>.

Resumen

- Título: Modelando la variabilidad: Métodos formales para la representación de líneas de productos software.
- Autor: Carlos D. Camacho González
- Supervisores: Dr. Luis Llana Díaz y Dr. Alberto Núñez Covarrubias.
- Institución: Universidad Complutense de Madrid, España.
- Centro: Facultad de Informática.
- Programa: Doctorado en Ingeniería Informática.

Justificación del trabajo de investigación. El desarrollo de aplicaciones software basado en líneas de productos (SPLs) permite a los desarrolladores modelar sistemas en función de la relación entre sus componentes. Así, es posible estudiar la variabilidad y características comunes entre ellos. Este paradigma de desarrollo se remonta a finales de los años 80. Desde entonces, se han realizado numerosos aportes sobre las distintas maneras de modelar las relaciones entre sus componentes [7]. Inicialmente, estas relaciones fueron definidas de forma gráfica, sirviendo como soporte para la toma de decisiones en las fases iniciales del diseño de los productos. Sin embargo, para detectar errores de diseño en fases tempranas del proceso de desarrollo y realizar comparaciones sobre los productos construidos, surge la necesidad de procesar y analizar automáticamente estos modelos. Por ello, en esta tesis se propone la definición de métodos formales para modelar SPLs, permitiendo automatizar su análisis y procesamiento.

Preguntas que motivan la investigación. De las condiciones descritas anteriormente, surgen las siguientes preguntas que motivan el desarrollo de esta tesis doctoral. ¿Es posible definir un marco formal que permita modelar FODA y sea extensible a otros modelos de variabilidad?, ¿Cómo analizar una SPL sin generar todos sus productos válidos?, ¿Cuál es el efecto de preprocesar los términos del álgebra antes de obtener los productos válidos?,

¿Cuáles son las ventajas de comparar productos en función de su coste?, ¿Es posible determinar la probabilidad de ocurrencia de una característica en un modelo cualquiera?. Una vez descritas estas preguntas se procede a definir el objetivo global y los objetivos específicos de la tesis doctoral.

Objetivos. El objetivo principal de esta tesis doctoral² es definir nuevos métodos formales para la representación de **SPLs**, teniendo en cuenta aquellos modelos que carecen de formalismos matemáticos para su representación. Partiendo de este objetivo general, se desarrollan los siguientes objetivos específicos: i) definir una representación formal para **FODA** que permita realizar análisis automatizado sobre su estructura; ii) definir el coste dentro del proceso de construcción de los productos válidos; iii) representar probabilidades dentro de los términos; y iv) mostrar distintas implementaciones de los formalismos descritos en función de la información que sea necesario generar.

Contribuciones. Esta tesis doctoral ha permitido definir un marco teórico para representar formalmente diagramas **FODA**, incluyendo su sintaxis, el proceso de traducción de **FODA** a **SPLA**, su conjunto de reglas semánticas, implementación y posibles casos prácticos. Seguidamente, dada la necesidad de representar costes en los modelos, se extendió la definición de **SPLA** para contemplar costes. De igual manera, se muestra la equivalencia de los dos modelos hasta ahora planteados, su implementación y posibles casos prácticos. Para finalizar las contribuciones de esta tesis doctoral, se planteó una nueva extensión para determinar e identificar aquellos componentes y configuraciones válidas, más o menos, frecuentes en la línea de productos.

Palabras clave: Líneas de productos software, FODA, Métodos formales, Álgebras de procesos, Semántica operacional, Semántica denotacional, Semántica axiomática.

²El contenido de esta tesis doctoral y todos sus soportes se encuentran disponibles en la siguiente URL: <http://ccamacho.github.io/phd/>.

Contenido

	Página
Abstract	I
Resumen	III
Lista de figuras	VII
1. Introducción	1
1.1. Motivación y alcance de esta tesis	2
1.2. Objetivos	5
1.3. Estructura del documento	7
2. Trabajos relacionados	9
2.1. Líneas de productos	10
2.2. Líneas de productos software	10
2.3. Fases de desarrollo de las SPLs	11
2.3.1. Diseño (<i>Domain Engineering</i>)	11
2.3.2. Implementación (<i>Application Engineering</i>)	11
2.4. Modelos de características (<i>Feature models</i>)	12
2.4.1. Análisis del dominio orientado a características (FODA)	12
2.5. Métodos Formales en SPLs	14
2.5.1. Álgebras de procesos	15
2.5.2. Sistemas de etiquetado de transiciones (LTS)	16
2.5.3. Sistemas de transiciones modales (MTS)	17
2.5.4. Sistemas de transiciones modales extendidos (EMTS)	18
2.5.5. Sistemas de transiciones modales extendidos generalizados (GEMTS)	19
2.5.6. Lógica proposicional	20
2.5.7. Modelos de características deónticos (MHML)	21
2.5.8. Autómatas modales de Entrada/Salida	21
2.5.9. Redes de Petri con características	22
2.5.10. Estimación del coste en SPLs	23
2.5.11. Líneas de productos y <i>SAT solvers</i>	24
2.5.12. Técnicas de optimización del tiempo de ejecución	24
2.5.13. Álgebras de procesos y análisis del coste	25
2.5.14. Probabilidades en álgebras de procesos	26

3. SPLA: Marco teórico para representar diagramas FODA	27
3.1. Sintaxis	28
3.2. Traducción de FODA a SPLA	31
3.3. Semántica Operacional	33
3.4. Semántica Denotacional	42
3.5. Reducción de los términos sintácticos	56
3.6. Semántica Axiomática	57
3.7. Consistencia en la traducción de FODA a SPLA	68
3.8. Factibilidad de los términos SPLA	70
3.9. Caso de estudio: Sistema VSS	83
3.10. Implementación de SPLA	90
3.10.1. Módulo de factibilidad	90
3.10.2. Módulo denotacional	90
4. SPLA^C: Extensión de SPLA para representar costes	93
4.1. Función de coste	94
4.2. Transiciones con coste	96
4.2.1. Nueva regla [req2]	99
4.2.2. Ejemplo de ejecución de reglas	102
4.3. Chef.io	106
4.4. Cálculo de coste en listas de ejecución para chef.io	109
4.4.1. Organización de la línea de productos	109
4.4.2. Especificación del coste de las características	113
4.4.3. Ejecución y resultados	116
4.5. Implementación	117
4.5.1. Consideraciones	117
5. SPLA^P: Extensión de SPLA para representar probabilidades	123
5.1. Sintaxis	124
5.2. Semántica operacional	124
5.3. Consistencia del modelo probabilístico	129
5.4. Semántica denotacional	131
5.5. Equivalencia entre la semántica operacional y la semántica denotacional . . .	137
5.6. Ocultando conjuntos de características	146
5.7. Implementación	149
6. Conclusiones	153
6.1. Contribuciones principales	154
6.1.1. SPLA: Marco teórico para representar diagramas FODA	154
6.1.2. SPLA ^C : Extensión de SPLA para representar costes	155
6.1.3. SPLA ^P : Extensión de SPLA para representar probabilidades	156
6.2. Publicaciones obtenidas con esta tesis doctoral	157
6.3. Trabajo futuro	158
Bibliografía	159

Lista de figuras y tablas

2.1.	Representación de las relaciones en FODA.	13
2.2.	Ejemplos de diagramas FODA.	13
2.3.	Ejemplo de PL-CCS [79].	16
2.4.	Sistema de etiquetado de transiciones [40].	17
2.5.	Sistemas de etiquetado de transiciones modales extendidos.	18
2.6.	Operadores de cardinalidad en sistemas de etiquetado de transiciones.	19
2.7.	Modelo de características y su equivalente en lógica proposicional [36].	20
2.8.	Red de características para la línea de productos $\{\{\text{Café}\}, \{\text{Café}, \text{Leche}\}\}$	22
2.9.	Sistemas de transiciones probabilísticos.	26
3.1.	Traducción de FODA a SPLA.	32
3.2.	Ejemplos de traducciones de diagramas FODA a términos SPLA.	33
3.3.	Reglas de la semántica operacional de SPLA.	34
3.4.	Aplicación de las reglas de la semántica operacional para los operadores [feat] , [ofeat] y [cho]	39
3.5.	Aplicación de las reglas de la semántica operacional para el operador [con]	40
3.6.	Aplicación de las reglas de la semántica operacional para los operadores [excl] y [mand]	41
3.7.	Aplicación de las reglas de la semántica denotacional.	44
3.8.	Axiomas para eliminar los operadores de <i>requerimiento</i> , <i>obligación</i> , <i>exclusión</i> y <i>prohibición</i>	57
3.9.	Axiomas para eliminar los operadores <i>básicos</i> , <i>características opcionales</i> y el operador de <i>conjunción</i>	58
3.10.	Transformación a forma normal para los ejemplos d , e y f	64
3.11.	Transformación a forma normal para el ejemplo g	65
3.12.	Representación FODA del <i>software</i> de <i>streaming</i> de video.	84
3.13.	Sistema de etiquetado de transiciones para VSS.	85
3.14.	Semántica denotacional para VSS.	87
3.15.	Reglas de deducción aplicadas a VSS (1 de 2).	88
3.16.	Reglas de deducción aplicadas a VSS (2 de 2).	89
3.17.	Benchmark de factibilidad.	91
3.18.	Benchmark denotacional.	92
4.1.	Ejemplo del modelado del coste en FODA (1 de 2).	103
4.2.	Ejemplo del modelado del coste en FODA (2 de 2).	103
4.3.	Función de coste	104
4.4.	Arquitectura general del servicio de <i>Chef</i>	107
4.5.	Modelo de características de chef.io.	113
4.6.	Traducción de diagrama FODA a SPLA.	114

4.7.	Término optimizado SPLA	115
4.8.	Tiempos de ejecución para 1, 2, 4 y 8 nodos y 1, 2, 4, 8, 16 y 32 hilos de ejecución.	120
4.9.	Tiempo acumulativo para el procesamiento de productos sin umbrales (izquierda) y con umbrales Min=1, Max=2 (derecha).	121
5.1.	Reglas para definir la semántica operacional de SPLA	125
5.2.	Ejemplos de la ejecución de la reglas de la semántica operacional del modelo probabilístico.	130
5.3.	Ejemplo de la ejecución de las reglas de la semántica denotacional (1 de 2). .	136
5.4.	Ejemplo de la ejecución de las reglas de la semántica denotacional (2 de 2). .	137
5.5.	Semántica operacional para el operador de ocultamiento.	146
5.6.	Tiempo de procesamiento para un modelo con 1500 características.	150
5.7.	Probabilidad de las características del modelo de variabilidad.	151

Capítulo 1

Introducción

“The best way to get a project done faster is to start sooner.”

Jim Highsmith

Contenido

1.1. Motivación y alcance de esta tesis	2
1.2. Objetivos	5
1.3. Estructura del documento	7

Las líneas de productos (PLs, del inglés Product Lines) permiten construir plataformas comunes para la creación de productos, reutilizando componentes previamente desarrollados [87]. Esta plataforma, la cual es común a todas las variantes de productos, permite construir, mantener, mejorar y gestionar, de manera eficiente el ciclo de vida de los productos.

En el campo de las PLs existen dos conceptos clave que permiten modelar productos en función de sus componentes comunes y variables. Estos conceptos, en inglés *commonality* y *variability* describen respectivamente, la organización de los bloques funcionales que serán comunes a todos los productos, así como aquellos que serán distintos en cada producto válido del modelo.

Sin embargo, deben existir reglas que permitan relacionar estos elementos entre sí. Así, surgen las relaciones entre los componentes de la línea de productos. Estas reglas permiten modelar relaciones entre componentes, de forma que puedan construirse productos válidos

y permitir la definición de comportamientos tales como los siguientes:

- Si existe el componente A entonces no puede incluirse el componente B.
- Si existe el componente A entonces debe incluirse el componente B.
- Si existe el componente A entonces deben incluirse los componentes B, C y D.
- Si existe el componente A entonces pueden incluirse los componentes B, C o D.
- Si existe el componente A entonces debe incluirse solo uno de los componentes B o C.

Todos los conceptos y herramientas de análisis para el modelado de PLS han sido aplicados directamente al desarrollo de software, dando lugar al término de líneas de productos software.

Las líneas de productos software (SPLs, del inglés Software Product Lines) surgen de forma similar a las líneas de productos tradicionales, esto es, por la necesidad de crear sistemas complejos que permitan satisfacer necesidades particulares de los usuarios y, a su vez, por la necesidad de poder gestionar su ciclo de vida de la manera más eficiente posible, reutilizando los recursos existentes [65, 33].

El presente capítulo describe la motivación, objetivos y alcances de esta tesis doctoral, los cuales se basan en modelar y dar un significado formal a las reglas descritas anteriormente. Gracias a la definición de estas reglas, es posible utilizar herramientas de análisis automatizado para generar información útil, tanto para la detección de errores en fases tempranas del diseño de los productos de la SPL, como para ayudar en procesos de toma de decisiones sobre el mantenimiento y la gestión del ciclo de vida de los productos.

1.1. Motivación y alcance de esta tesis

En la fase de diseño de las SPLs, los modelos de variabilidad son representados de forma gráfica, lo cual representa un problema para analizarlos de forma automatizada [66, 62]. De esta forma, surge la necesidad de dotar de una representación formal a los modelos de variabilidad que actualmente carecen de métodos formales para su definición [35, 21, 95].

Para el desarrollo de esta tesis doctoral se ha elegido estudiar el análisis del dominio orientado a características (**FODA**, del inglés Feature Oriented Domain Analysis), como modelo de variabilidad a analizar, ya que es ampliamente utilizado en la industria y utiliza una representación gráfica para su estudio [33, 5].

En función de la necesidad planteada, surgen las siguientes preguntas que motivan el desarrollo de esta tesis:

- ¿Es posible definir un marco formal que permita modelar **FODA** y sea extensible a otros modelos de variabilidad?

Partiendo del argumento que **FODA** es un modelo ampliamente extendido en el estudio de las **PLs**, se busca establecer como objetivo modelarlo formalmente, para así permitir que cualquier estudio de **PLs** o **SPLs** pueda beneficiarse de los resultados obtenidos en este trabajo. Además, es necesario extender el conjunto de relaciones, así como la información descrita en el modelo, para complementar el análisis con modelos de coste y probabilidades, todo esto, describiendo el modelo de tal manera que puedan agregarse más reglas semánticas en el futuro.

- ¿Cómo analizar una **SPL** sin generar todos sus productos válidos?

Generar todas las posibles combinaciones de productos (válidos o no) de una **SPL**, supone un problema combinatorio complejo de resolver dado su alto coste computacional. Parte de la motivación de este trabajo surge de la necesidad de definir métodos que permitan proporcionar información útil, sin generar todos los productos válidos de una **SPL**.

- ¿Cuál es el efecto de preprocesar los términos del álgebra antes de obtener los productos válidos?

En el caso particular de este trabajo, existen operadores en la descripción del álgebra que incrementan exponencialmente el número de combinaciones válidas. Debido a que no siempre son necesarias todas las combinaciones de productos, es interesante

describir el comportamiento de los algoritmos que analizan y preprocesan los modelos para reducir, en la medida de lo posible, el tiempo de cómputo de los mismos.

- ¿Cuáles son las ventajas de comparar productos en función de su coste?

Los modelos de variabilidad permiten describir productos en función de sus características. Así, productos distintos entre sí, podrían estar compuestos por las mismas características. De esta forma, los productos **AB** y **BA**, estando ambos compuestos por los mismos componentes, podrían ser distintos, por ejemplo, en su coste de producción. Esta condición es particularmente interesante, ya que tiene dos ventajas principales. La primera consiste en la posibilidad de comparar los productos a medida que se agregan características a los mismos. Por ejemplo, si un producto P , a medida que se va construyendo, sobrepasa un coste C , éste puede no ser válido y evitaría el procesamiento del término restante, reduciendo así tiempo de cómputo. La segunda ventaja consiste en poder determinar, de manera óptima, la construcción de los productos en función de su coste de producción. El orden en que se agregan los componentes al producto podría determinar que un producto requiera más tiempo en poder construirse, o que su coste de producción sea mayor.

- ¿Es posible determinar la probabilidad de ocurrencia de una característica en un modelo cualquiera?

La gestión del ciclo de vida de las **SPLs** incluye, por ejemplo, la desincorporación de las características que puedan estar en desuso [67]. Por ello, es necesario determinar la probabilidad de ocurrencia de las características que componen el modelo de variabilidad. Así, se podrá proveer de herramientas de análisis automatizado a los procesos relacionados con la toma de decisiones en la gestión del ciclo de vida de las **SPLs**.

A partir del planteamiento de estas preguntas de investigación, se han definido los objetivos de esta tesis doctoral.

1.2. Objetivos

El objetivo principal de esta tesis doctoral consiste en **definir nuevos métodos formales para representar SPLs. En particular, dotar a FODA de un formalismo que permita realizar análisis automático sobre sus estructuras gráficas, sin restringir su aplicación a otras metodologías.** De esta manera, para alcanzar el objetivo principal, se han definido los siguientes objetivos específicos:

- ① **Definir una representación formal para FODA que permita realizar análisis automatizado sobre su estructura.** Este formalismo debe contener tanto la definición de la sintaxis del álgebra, como un mecanismo de traducción de FODA a la nueva representación. Una vez traducidas estas estructuras no formales, se deberá mostrar la definición de las reglas semánticas¹ que permitan procesar los términos del álgebra. Dado que existen distintas maneras de procesar los términos, puesto que hay distintas semánticas, es necesario demostrar que las representaciones son equivalentes y correctas. Es importante destacar que partiendo de la semántica operacional² pueden generarse productos válidos y distintos entre sí, que serán equivalentes en la semántica denotacional³. Además, surge la necesidad de buscar estructuras más simples que permitan describir los mismos modelos utilizando un número menor de elementos sintácticos. Para ello es necesario definir las formas normales y pre-normales del álgebra, así como demostrar, formalmente, que los modelos resultantes son equivalentes al original en cuanto a los productos que genera.

- ② **Definir el coste dentro del proceso de construcción de los productos válidos.** El modelado del coste en el álgebra se desarrollará con la finalidad de optimizar el procesamiento de los términos. De esta forma, cuando un producto cumpla una condición determinada, podrán definirse acciones tales como dejar de procesar el término

¹Semántica operacional, semántica denotacional y semántica axiomática.

²El orden en que se generan las características **sí** es importante dentro de la construcción del producto.

³El orden en que se generan las características **no** es importante dentro de la construcción del producto.

cuando el producto supere un umbral de coste máximo, o considerar únicamente como productos válidos aquellos cuyo coste se encuentre en un rango de valores previamente definido. Además, modelar el coste de los productos permitirá realizar comparaciones entre diferentes productos válidos para determinar, por ejemplo, cuál de ellos conlleva un menor, o mayor, coste de producción.

- ③ **Representar probabilidades dentro de los términos.** Utilizar probabilidades en el modelo permite determinar qué componente o producto es el más utilizado entre los distintos productos válidos. Utilizando esta representación, es posible tomar acciones tales como asignar de manera eficiente recursos al ejecutar pruebas unitarias, descatalogando así componentes en desuso dentro de la SPL. De esta manera, al seleccionar una característica para calcular su probabilidad dentro del término, se podría permitir que todas aquellas características, a excepción de la que se está estudiando, sean llamadas de la misma manera. Esto permitiría centrar el estudio en los elementos sintácticos que afecten la cardinalidad de los productos válidos generados y, de esta manera, aumentar el desempeño de los algoritmos que calculen los valores de estas probabilidades. Este objetivo plantea ventajas directamente relacionadas a la toma de decisiones en la gestión del ciclo de vida de las SPLs, ya que permite determinar qué componentes o productos son más o menos utilizados.
- ④ **Mostrar distintas implementaciones del formalismo descrito, en función de la información que sea necesario generar.** Una vez definidos los aspectos formales del álgebra, se debe proceder a su implementación, o implementaciones, dependiendo de la información que sea necesario generar. Consecuentemente, surge la necesidad de mostrar alternativas para el procesamiento de los términos del álgebra. Por ejemplo, soluciones de tipo *greedy* permiten calcular todos los productos válidos de la SPL, teniendo en cuenta problema de la explosión combinatoria en su procesamiento. Para este tipo de problemas es necesario analizar y optimizar el término antes de procesarlo y, en medida de lo posible, analizar los términos del álgebra en sistemas distribuidos de

forma paralela. También es posible determinar si un término genera productos válidos o no mediante el uso de *SAT solvers*. En este caso, debe ser descrito previamente el proceso de traducción de un término del álgebra a una expresión lógica. De igual manera, es posible realizar implementaciones de tipo *branch and bound* donde, al calcular el coste de producción de los productos de la SPL es posible definir umbrales, y así optimizar el desempeño del algoritmo evitando procesar términos que no se encuentren acotados por los mismos. Implementaciones que permitan realizar análisis probabilístico son de igual manera interesantes, ya que gracias a estas, podremos identificar los componentes que son más o menos utilizados en los productos.

Todos los problemas y objetivos planteados en esta sección están descritos en detalle en el desarrollo de este trabajo. Adicionalmente, cada problema estará acompañado de un caso de uso práctico y una implementación acorde a su descripción formal. De manera general, esta tesis doctoral se centra en el planteamiento de nuevos métodos formales para representar y analizar SPLs, mostrando distintas implementaciones de acuerdo a la información que sea necesario generar.

1.3. Estructura del documento

Capítulo 2. Describe los trabajos previos relacionados con las SPLs, así como distintas maneras de representarlas, tanto gráficamente como formalmente.

Capítulo 3. Muestra la descripción general del álgebra SPLA junto con su sintaxis. Además, se presentan tres semánticas que darán significado a los elementos sintácticos descritos anteriormente, la semántica operacional, la semántica denotacional y la semántica axiomática. Este capítulo incluirá un caso de uso práctico para modelar los componentes de un sistema de *streaming* de video utilizando SPLs. En este capítulo también se mostrará la implementación de la semántica denotacional, así como un módulo de satisfactibilidad para comprobar si un modelo cualquiera genera productos válidos. Este capítulo cubre los objetivos ① y ④.

Capítulo 4. Describe la extensión de SPLA relacionada con el modelado de costes. Este capítulo presenta la descripción formal que permite al álgebra realizar el procesamiento del coste de los productos a medida que se computan los términos. En este capítulo se presentan ejemplos sobre el cómputo de las reglas, así como del análisis de distintos mecanismos para agilizar la ejecución de los algoritmos descritos. Además, se incluye la descripción de los componentes de un sistema de gestión de configuración (*Chef.io*), utilizado en el ejemplo práctico para la implementación de una SPL. Este capítulo cubre los objetivos ② y ④.

Capítulo 5. Detalla la extensión de SPLA que permite el cálculo de la probabilidad de ocurrencia de las características dentro del modelo. Este capítulo incluye una extensión de la implementación de la semántica denotacional, donde se calcula la probabilidad P de que una característica se encuentre en un término T . Este capítulo cubre los objetivos ③ y ④.

Capítulo 6. El último capítulo describe las conclusiones, contribuciones y trabajo futuro de esta tesis doctoral.

Capítulo 2

Trabajos relacionados

“Research is to see what everybody else has seen, and to think what nobody else has thought.”

Albert Szent-Gyorgy

Contenido

2.1. Líneas de productos	10
2.2. Líneas de productos software	10
2.3. Fases de desarrollo de las SPLs	11
2.3.1. Diseño (<i>Domain Engineering</i>)	11
2.3.2. Implementación (<i>Application Engineering</i>)	11
2.4. Modelos de características (<i>Feature models</i>)	12
2.4.1. Análisis del dominio orientado a características (FODA)	12
2.5. Métodos Formales en SPLs	14
2.5.1. Álgebras de procesos	15
2.5.2. Sistemas de etiquetado de transiciones (LTS)	16
2.5.3. Sistemas de transiciones modales (MTS)	17
2.5.4. Sistemas de transiciones modales extendidos (EMTS)	18
2.5.5. Sistemas de transiciones modales extendidos generalizados (GEMTS)	19
2.5.6. Lógica proposicional	20
2.5.7. Modelos de características deónticos (MHML)	21
2.5.8. Autómatas modales de Entrada/Salida	21
2.5.9. Redes de Petri con características	22
2.5.10. Estimación del coste en SPLs	23
2.5.11. Líneas de productos y <i>SAT solvers</i>	24
2.5.12. Técnicas de optimización del tiempo de ejecución	24
2.5.13. Álgebras de procesos y análisis del coste	25
2.5.14. Probabilidades en álgebras de procesos	26

Este capítulo presenta el estado del arte en el campo de las **SPLs**. En particular, muestra los trabajos relacionados con los métodos formales para definir modelos de variabilidad, describiendo las distintas técnicas empleadas actualmente para el estudio y análisis automatizado de las mismas.

2.1. Líneas de productos

Las líneas de productos hacen referencia a la agrupación de un conjunto de activos dentro de las compañías, estos activos son caracterizados en función de sus componentes variables y comunes [26, 102]. Para la producción de productos de forma masiva, es habitual tener una plataforma común, y a partir de ella, desarrollar un conjunto de módulos para satisfacer las necesidades particulares de los usuarios [44]. Un ejemplo claro de ello, en el ámbito de la construcción de vehículos, es la plataforma de construcción transversal modular (MQB, del alemán Modularer Querbaukasten) del grupo Volkswagen, que permite desarrollar múltiples modelos de vehículos para marcas pertenecientes al grupo Alemán [97]. En este caso particular, existe una serie de componentes mecánicos comunes, como lo son principalmente el chasis, motor y caja de cambios. Esto permite, entre otras cosas, el abaratamiento del coste de producción, tiempo de salida al mercado y la reducción del tiempo invertido en probar cada uno de los componentes del vehículo producido, ya que en este caso hay una serie de componentes comunes que sólo deben ser probados individualmente, una vez [33].

Este paradigma de desarrollo también permite el estudio localizado de los componentes para determinar su beneficio, frecuencia de aparición y si deben ser, o no, creados, mantenidos o desechados.

2.2. Líneas de productos software

Las líneas de productos software (**SPLs**, del inglés Software Product Lines) representan una forma sistematizada y estructurada para construir y mantener los soportes lógicos de los sistemas informáticos [7, 33, 12, 65]. La finalidad de este paradigma consiste en optimizar

el coste de operación y de creación de productos nuevos [14]. De igual manera, al reutilizar componentes, se incrementa la calidad de los productos recién desarrollados ya que se están reutilizando módulos que han sido previamente comprobados en otros productos [68, 88].

El Instituto de Ingeniería de Software (SEI, del inglés Software Engineering Institute), define una línea de productos software como: “*A set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*” [76, 99, 33].

2.3. Fases de desarrollo de las SPLs

El desarrollo de SPLs se define en dos fases, diseño e implementación [65]. En estas fases está contenida la definición del ciclo de vida de la línea de productos, su implementación y descripción del proceso de mejora continua [60].

2.3.1. Diseño (*Domain Engineering*)

Esta fase consiste en identificar el dominio en el cual va a desarrollarse la SPL, su objetivo y para qué va a ser utilizada [98]. Generalmente, el proceso de diseño consiste en identificar tanto los componentes que difieren entre sí, como los que son comunes entre cada producto desarrollado, así como también, realizar el proceso de recolección de los requerimientos del sistema [89]. Esta información debe ser modelada de acuerdo al modelo de variabilidad a utilizar, y en función de esto, planificar cómo serán construidos los productos [48].

2.3.2. Implementación (*Application Engineering*)

La fase de implementación se centra en la construcción de los productos pertenecientes a la SPL, maximizando, en la medida de lo posible, la reutilización de los componentes disponibles [7, 65, 60]. Básicamente, en esta fase se traducen los componentes de la fase de diseño a componentes funcionales, los cuales serán parte de la línea de productos. Además,

en esta fase se describen los procesos referentes a la documentación de la interacción entre los componentes de la fase de diseño, la arquitectura y casos de prueba. Para ello, se emplean herramientas automatizadas que permitan detectar tanto errores en los modelos, como detectar las características que nunca aparecerán en productos válidos. Estas herramientas también permiten generar información sobre los modelos ya definidos como, por ejemplo, el número total y coste de los productos válidos, entre otros [102].

2.4. Modelos de características (*Feature models*)

Para las distintas fases del proceso del desarrollo de SPLs se presenta un modelo común. Los modelos de características son representaciones que permiten modelar la organización de los componentes que forman las SPLs. Estos modelos permiten definir los componentes funcionales y no funcionales en una misma estructura, así como modelar su interacción [92]. Además, éstos son especificados, generalmente, de manera jerárquica, utilizando grafos donde los vértices representan puntos de variación y las aristas relaciones entre los componentes [98, 38].

Los modelos de características permiten representar propiedades como por ejemplo:

Existe un producto con las características A y C.

Adicionalmente, dentro de los modelos de características pueden representarse restricciones sobre sus componentes. Por ejemplo, podría ser representada la siguiente propiedad:

En cualquier producto válido, si se incluye la característica C, entonces las características A y B deben ser incluidas de igual manera.

2.4.1. Análisis del dominio orientado a características (FODA)

El análisis del dominio orientado a características (FODA, del inglés Feature Oriented Domain Analysis), es una metodología cuyo origen surge del estudio de diversos métodos de diseño de las SPLs [61]. Esta metodología se centra en la descripción de aquellos elementos

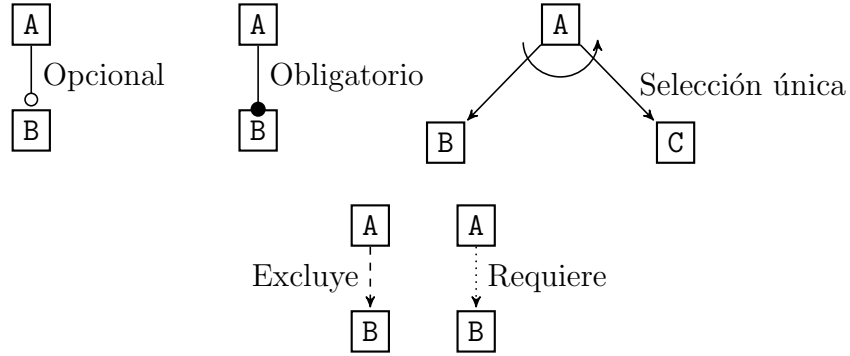


Figura 2.1: Representación de las relaciones en FODA.

variables y comunes dentro del modelo, permitiendo generar una representación gráfica de las relaciones entre los componentes. En este modelo los nodos representan componentes, mientras que las conexiones entre ellos representan sus relaciones [25]. La figura 2.1 muestra la descripción gráfica de las relaciones de este modelo.

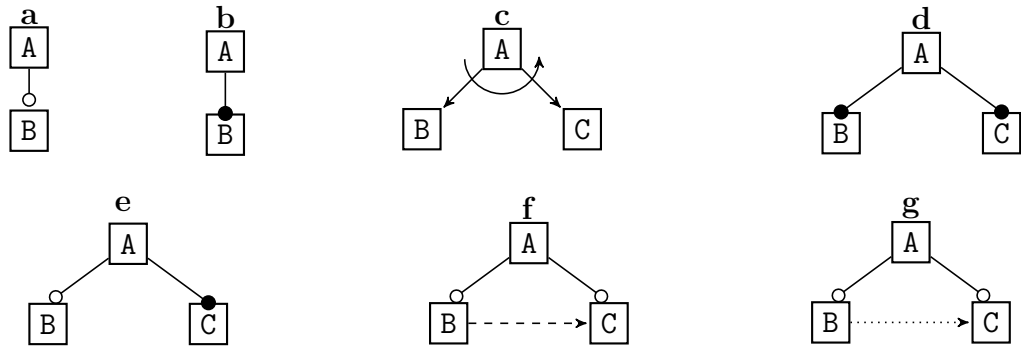


Figura 2.2: Ejemplos de diagramas FODA.

Utilizando como referencia los diagramas FODA de la figura 2.2, los ejemplos **a** y **b** muestran dos SPLs con dos posibles características A y B. Con respecto al ejemplo **a**, la característica A aparecerá en todos los productos válidos del modelo, mientras que B será opcional. De esta manera, los productos válidos generados por este modelo serán, por un lado, aquellos

con la característica **A** y, por otro lado, aquellos con las características **A** y **B**. En el ejemplo **b** ambas características son obligatorias, ya que cualquier producto generado de esta SPL contendrá **A** y **B**.

El ejemplo **c** representa una SPL con el operador de *selección única*. En este ejemplo hay tres características diferentes: **A**, **B** y **C**. Cualquier producto de **c** contendrá **A** y una de las dos características restantes, **B** o **C**. El operador de *paralelo* se muestra en los ejemplos **d** y **e**. En ambos casos, la característica **A** estará en todos los productos. Las ramas del ejemplo **d** son obligatorias, por lo cual sólo se generará un producto, aquel que contenga **A**, **B** y **C**. En el ejemplo **e**, una rama es opcional y la otra obligatoria, con lo cual se generarán dos productos en el modelo, uno con las características **A** y **C**, y otro con las características **A**, **B** y **C**.

Finalmente, propiedades mas complejas aparecen en los ejemplos **f** y **g**. En estos diagramas existen restricciones combinadas con características opcionales. En el ejemplo **f** existe una restricción de *exclusión*, esto es, si **B** es incluida en un producto, entonces **C** no puede estar en el mismo producto. Por el contrario, en el ejemplo **g** existe una restricción de *obligatoriedad*, esto es, si **B** es incluida, entonces **C** también debe ser incluida.

2.5. Métodos Formales en SPLs

Los métodos formales en SPLs surgen de aplicar formalismos matemáticos a modelos de variabilidad, de forma que pueda comprobarse, de manera automatizada, si su especificación es equivalente a su implementación [4, 17, 36, 47, 46, 50, 9, 22, 91, 77, 11, 100, 59, 90].

El estudio formal de las líneas de productos puede clasificarse en dos fases, de acuerdo a la información que sea generada [39]. La primera fase consiste en definir matemáticamente la sintaxis y las semánticas con las que sea representado el modelo formal. La segunda fase consiste en desarrollar la técnica empleada para generar conocimiento a partir de las estructuras previamente definidas [57, 32, 63, 76].

En esta sección se incluyen los *frameworks* formales usados para describir SPLs, así como

los modelos de variabilidad. Estos *frameworks* describen la sintaxis y la semántica de las distintas álgebras descritas. La especificación de los sistemas incluyen distintos aspectos, tales como describir su comportamiento funcional o no funcional, así como su estructura o arquitectura [16, 15, 43, 93].

2.5.1. Álgebras de procesos

La descripción del comportamiento del sistema requiere, para poder ser expresada formalmente, de un conjunto de reglas que especifiquen los estados y transiciones al cambiar de un estado a otro [80]. El procesamiento de un conjunto de estados y reglas de transición se define como un *proceso*. Estos procesos pueden ser ejecutados de manera secuencial, que implicará que los procesos se ejecutarán uno tras otro, o pueden ser ejecutados en paralelo, lo que quiere decir que se ejecutarán de manera concurrente. De esta forma todos los procesos se ejecutarán al mismo tiempo y permitirá que el comportamiento de cualquier proceso pueda tener influencia en los otros [83, 101, 51, 29].

El modelado del comportamiento de los procesos se basa en el estudio de cómo los datos son manipulados entre ellos. Este modelado incluye la definición del mecanismo de control que especifica cómo los sistemas intercambian y manipulan los datos. Los procesos son dinámicos y activos, sin embargo, los datos en sí representan una estructura estática y pasiva. El comportamiento del sistema es definido como la ejecución de varios procesos concurrentes, donde estos procesos pueden intercambiar datos para influenciar el comportamiento de otros procesos.

El álgebra de procesos basado en el cálculo de sistemas comunicantes (PL-CCS, del inglés Product Lines CCS) ha sido descrito como un álgebra basado en el clásico estudio del cálculo de sistemas comunicantes (CCS, del inglés Calculus of Communicating Systems), extendiendo el modelo original representando modelos de variabilidad [47]. En este modelo, la variabilidad del sistema es descrita de la misma forma en que es modelado su comportamiento. Para ello se utilizan operadores de paralelo, selección binaria, opcional, elección

no determinista, restricción y nulo. También, se describe un sistema de etiquetado de transiciones para describir las acciones de comunicación y las relaciones de transición para la semántica definida. Es importante destacar que este formalismo no está relacionado con ningún modelo de variabilidad como FODA, RSEB o PLUSS. En la figura 2.3 se muestra un ejemplo del álgebra PL-CCS [79]. En este ejemplo, se describe cómo la variabilidad es representada utilizando álgebras de procesos. Las relaciones se utilizan para describir decisiones de configuración entre un conjunto de características.

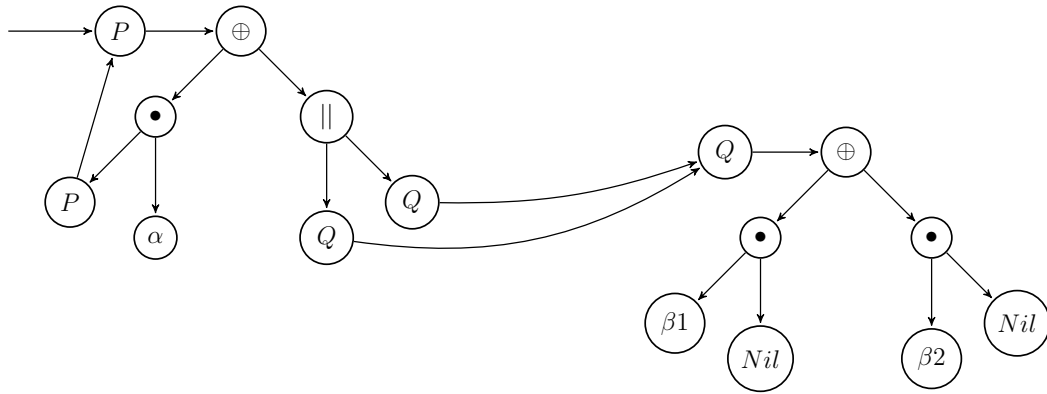


Figura 2.3: Ejemplo de PL-CCS [79].

2.5.2. Sistemas de etiquetado de transiciones (LTS)

Los sistemas de etiquetado de transiciones tradicionales (LTSs, del inglés Labelled Transition Systems) son extendidos y adaptados para representar SPLs [84, 41, 39, 69, 73, 40, 10, 9]. Un LTS puede ser descrito como la representación del comportamiento de un sistema donde existen eventos que cambian el comportamiento del mismo. Cuando un LTS representa una SPL, los eventos suelen referirse a las características [70]. Es importante destacar que los LTS son formalismos ampliamente estudiados en la literatura.

En la figura 2.4 se muestra un ejemplo básico de una máquina expendedora de café, té y refrescos. En esta máquina, se representan todos los estados, en los cuales puede ocurrir

una transición de un estado a otro. Nótese que pueden verse cómo las transiciones entre los estados representan características, así como los estados representan puntos de variación dentro del dominio del sistema.

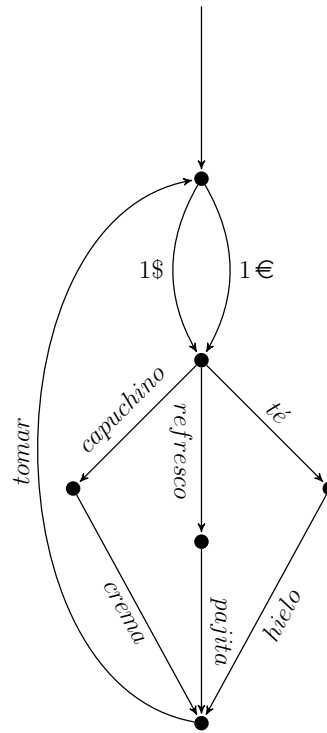


Figura 2.4: Sistema de etiquetado de transiciones [40].

2.5.3. Sistemas de transiciones modales (MTS)

Los sistemas de transiciones modales (MTSs, del inglés Modelled Transition Systems), se presentan como una extensión de los sistemas de etiquetado de transiciones tradicionales, en los que se presentan dos nuevos tipos de transiciones, transiciones obligatorias y transiciones opcionales [74, 6, 86]. Estos modelos son utilizados para describir el comportamiento de los sistemas cuando no se dispone de toda la información sobre el comportamiento del mismo. En este caso los sistemas de etiquetado de transiciones tradicionales dejan de ser válidos para estos modelos. Es interesante modelar sistemas parcialmente definidos ya que en las

etapas iniciales del desarrollo del *software*, no siempre puede definirse el comportamiento del sistema completo.

En el caso de los modelos de variabilidad basados en sistemas de etiquetado modales, gracias a la noción de incertidumbre descrita por la transición opcional, se podrán describir y diferenciar distintos comportamientos válidos dentro de un mismo modelo.

El desarrollo de SPLs está relacionado directamente con definir modelos de variabilidad en el ámbito del desarrollo de *software*. Estos describen cómo, al utilizar sistemas de etiquetado de transiciones, pueden modelarse transiciones obligatorias y opcionales para describir SPLs [72, 41, 86]. En estos trabajos se describen las semánticas para definir el comportamiento de los sistemas junto con sus ventajas y limitaciones, como por ejemplo, la falta de información adicional, *meta-datos* o atributos necesarios para definir el comportamiento del sistema.

2.5.4. Sistemas de transiciones modales extendidos (EMTS)

Los sistemas de etiquetado de transiciones tradicionales representan modelos donde, visto como un grafo, cada arista es identificada con la entrada que inicia una transición. Sin embargo, en las transiciones no se almacena el estado de la transición [45, 73]. Por ello surge la necesidad de extender el modelo inicial para agregar condiciones adicionales a estas transiciones, permitiendo crear transiciones obligatorias u opcionales en el modelo.

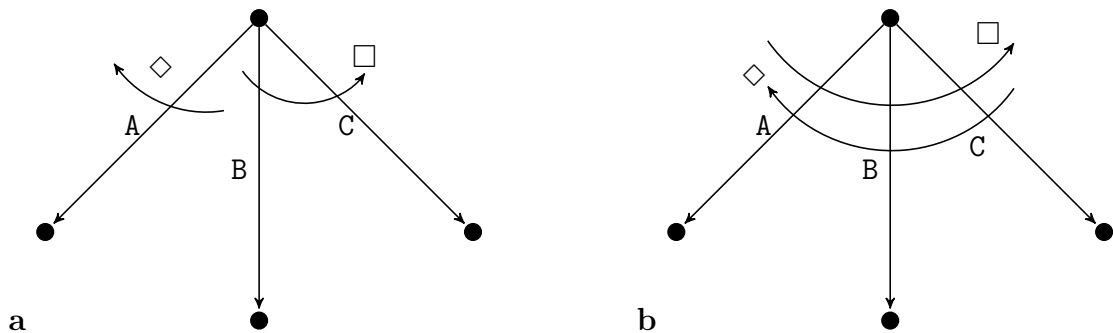


Figura 2.5: Sistemas de etiquetado de transiciones modales extendidos.

Si bien, la existencia de características opcionales y obligatorias es, en cierta manera útil, el objetivo es agregar más relaciones que extiendan y hagan el modelo inicial más flexible. Los sistemas de etiquetado de transiciones modales extendidos describen precisamente esta nueva abstracción, la cual permite crear transiciones con cardinalidad, es decir, permite definir condiciones de *al menos 1 de k ó hasta 1 entre k* características dentro del modelo de variabilidad [39].

En la figura 2.5, dos diagramas EMTS son descritos, para los cuales la relación *al menos 1 de n*, denotada con el símbolo \square , y la relación *hasta 1 de n*, representada con el símbolo \diamond , serán utilizadas para mostrar su comportamiento. El ejemplo **a** presenta un modelo en el que debe ser incluida al menos una característica, elegida entre B y C, en todos los productos, mientras que la característica A es opcional. En el ejemplo **b**, como A, B y C están relacionadas con el símbolo \square y \diamond , lo que quiere decir, que sólo puede ser seleccionada *exactamente* una característica en cada producto.

2.5.5. Sistemas de transiciones modales extendidos generalizados (GEMTS)

Con los GEMTS se agrega al modelo de variabilidad basado en LTSs el máximo grado de flexibilidad, ya que los operadores sintácticos permiten no sólo tomar *1 de n* características por operador, sino que serán *k de n* características las que puedan ser utilizadas [40].

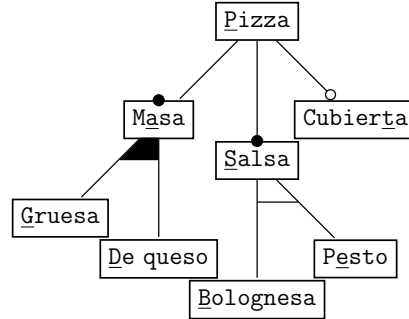
Operador sintáctico	LTS	MTS	EMTS	GEMTS
Hasta (\diamond)	No	<i>1 de 1</i>	<i>1 de n</i>	<i>k de n</i>
Al menos (\square)	No	<i>1 de 1</i>	<i>1 de n</i>	<i>k de n</i>

Tabla 2.6: Operadores de cardinalidad en sistemas de etiquetado de transiciones.

En la tabla 2.6 se puede observar la comparación sobre cómo se modela la cardinalidad en los distintos estudios relacionados a los LTSs.

2.5.6. Lógica proposicional

Una forma de representar modelos de variabilidad haciendo uso de métodos formales, consiste en procesar familias de productos utilizando lógica proposicional, donde los modelos de características se representan con una fórmula lógica, equivalente a sus elementos sintácticos [75, 36]. En términos matemáticos, un *álgebra* consiste en un conjunto de símbolos que denotan *valores* de algún tipo y *operaciones* sobre estos valores. En estos modelos, los operadores lógicos son utilizados sobre los operandos que representaran características. Estos trabajos emplean estructuras básicas extraídas de los modelos de características, como el operador o relación de obligatoriedad, opcional e implicación. De esta forma, estas estructuras son desarrolladas para reutilizar las herramientas existentes de análisis automatizado, tales como *SAT solvers* y bibliotecas para representar diagramas de decisión binaria (BDD, del inglés Binary-Decision Diagrams) [78, 52, 53, 47, 37].



Fórmula proposicional

F=

<i>hijo – padre :</i>	$(A \rightarrow P) \wedge (S \rightarrow P) \wedge (T \rightarrow P) \wedge$ $(G \rightarrow A) \wedge (D \rightarrow A) \wedge$ $(B \rightarrow S) \wedge (E \rightarrow S) \wedge$
<i>obligatoria :</i>	$(P \rightarrow A) \wedge (P \rightarrow S) \wedge$
<i>grupo – o :</i>	$(A \rightarrow G \vee D) \wedge$
<i>grupo – o – exclusivo :</i>	$(S \rightarrow B \vee E)$

Figura 2.7: Modelo de características y su equivalente en lógica proposicional [36].

La figura 2.7 muestra cómo un modelo de características es traducido a su equivalente en lógica proposicional. Sin embargo, existen trabajos relacionados con el estudio del proceso inverso, el cual consiste en inducir modelos de características tomando como punto de partida fórmulas proposicionales [36].

2.5.7. Modelos de características deónticos (MHML)

Los sistemas de etiquetado de transiciones tradicionales se utilizan como base para describir, haciendo uso de operadores deónticos básicos, modelos de características. Estos operadores deónticos, *es obligatorio que* o *está permitido que*, se utilizan junto a operadores básicos de la lógica proposicional tales como la negación, conjunción, disyunción e implicación, para modelar familias de productos[8].

La fórmula 2.1 describe un ejemplo de cómo usar el formalismo planteado.

$$(P (b) \text{ verdadero } \wedge (\epsilon \implies < c > \text{ verdadero})) \quad (2.1)$$

El significado de la fórmula 2.1 es el siguiente, se representa a un sistema cuyo estado actual permite la ejecución de la acción b (transición opcional) sin importar la acción de ϵ , en donde ejecutar la acción c será obligatorio.

Los formalismos anteriores sólo contaban con dos operadores sintácticos, *es obligatorio que* o *está permitido que*, mientras que ahora se dispone de todos los operadores de la lógica proposicional.

2.5.8. Autómatas modales de Entrada/Salida

Los autómatas de E/S son modelos que permiten modelar interfaces de comunicación entre sus componentes [19, 28, 3, 20]. De igual manera, un autómata de E/S puede ser descrito como subconjuntos de las características presentadas por los sistemas de modelado de transiciones de Larsen y Thomsen, extendiendo las características de los autómatas con interfaces [74, 69].

Todo el formalismo de los autómatas modales de E/S es aplicado directamente al modelado de familias de productos. Usualmente, los modelos de variabilidad son presentados en forma de árbol y éste es descrito como un todo, es decir, el árbol representa todos los productos distintos que pueden llegar a construirse. Los autómatas modales de E/S basan su estudio en la descomposición de este árbol, es decir, en subfamilias de productos. Seleccionar una característica de estas subfamilias para formar parte del producto en construcción se denomina *proceso de configuración*. Así, las subfamilias son modeladas como autómatas modales de E/S, los cuales reciben el nombre de modelos de variabilidad. Construir productos se define como el proceso de la composición de modelos de variabilidad, sí y sólo sí para éstos su entrada, salida y acciones ocultas, no se sobreponen. Así mismo, dos o más subfamilias pueden ser utilizadas para componer un modelo de más alto nivel.

2.5.9. Redes de Petri con características

Unos de los formalismos existentes para modelar familias de productos consiste en utilizar redes de Petri. Las redes de Petri se basan en modelar nodos que hacen referencia a lugares y transiciones, unidos por aristas. Éstas pueden ser extendidas para tener la capacidad de modelar familias de productos [81]. En este caso, las aristas modelan fórmulas proposicionales que representan características, de forma que cuando esta fórmula resulte *falsa*, la arista deja de existir.

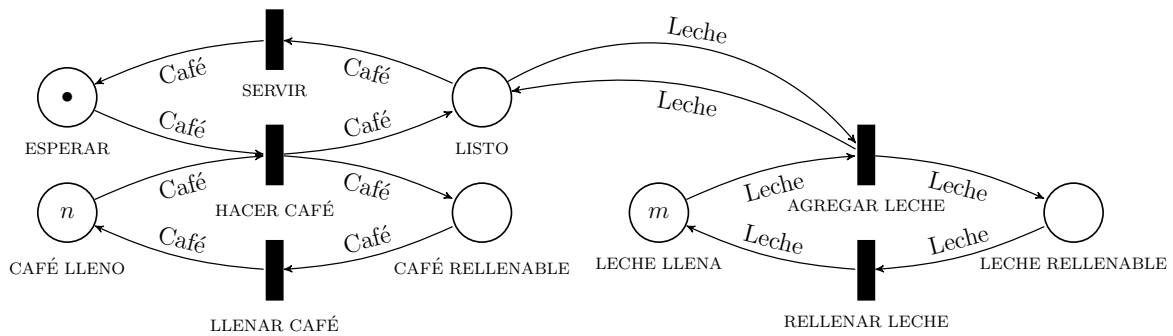


Figura 2.8: Red de características para la línea de productos $\{\{\text{Café}\}, \{\text{Café}, \text{Leche}\}\}$.

La figura 2.8 presenta un ejemplo de un modelo de variabilidad para modelar una máquina expendedora de café con dispensador de leche, considerada como una línea de productos con las características **Café** y **Leche**, donde el **Café** es obligatorio y la **Leche** opcional. En este caso, al ser *verdadero* el valor de la arista, se modifica el comportamiento del modelo, lo cual no ocurre en caso contrario.

2.5.10. Estimacion del coste en SPLs

Un factor importante en el desarrollo de *software* depende de la eficiencia y eficacia al estimar el coste del proyecto. Generalmente, el termino *coste* en SPLs se enfoca en cuantificar el esfuerzo que implica generar los productos válidos de una familia de productos. Para conseguir esto, todos los componentes del sistema deben ser cuantificados. Por ejemplo, el coste puede representar tiempo de desarrollo, el número de líneas de código o la dificultad de integrar un módulo a la línea de productos.

La estimación del coste puede estar basada en el tamaño del código o en métricas funcionales [82, 24, 58, 42, 55]. El término *tamaño* debe ser catalogado como un atributo importante de un producto *software* y no debe ser referenciado como su tamaño ocupado en memoria o su numero total de líneas de código. Adicionalmente, puede hacer referencia a la estimación acumulada de distintos aspectos, como la complejidad del problema a resolver, el número de desarrolladores necesarios, *hardware* necesario, e incluso estimar la complejidad de integrar los distintos componentes del *software* desarrollado. Este esfuerzo puede ser representado formalmente dependiendo de las necesidades del proyecto o requerimientos organizacionales [23]. Este tipo de análisis está basado en las convicciones de la gerencia del proyecto, y por lo tanto, no se consideran como métodos maduros dentro de la ingeniería de procesos. Otros estudios miden el esfuerzo analizando la evolución del ciclo del producto para así poder determinar su coste [94].

2.5.11. Líneas de productos y *SAT solvers*

Los métodos de análisis basados en el uso de *SAT solvers* han sido estudiados de manera extensiva para validar modelos de características [5, 93, 35, 17]. Estas soluciones se basan en definir un problema de factibilidad lógica, representando el modelo de variabilidad como una serie de condiciones lógicas. Una de las ventajas más relevantes de este tipo de aproximaciones es la posibilidad de utilizar *SAT solvers* para procesar modelos del orden de miles de características en el orden de milisegundos [4]. Estos métodos son utilizados para determinar si los modelos de variabilidad son factibles o no, lo que quiere decir, si son capaces de generar productos válidos. En este caso se han alcanzado tiempos de cómputo aceptables para modelos del orden de miles de productos válidos.

En contraste con estos modelos, el trabajo presentado en esta tesis doctoral propone varias mejoras. Por un lado, procesar el peor caso posible, donde se computan todos los productos de la SPL teniendo en cuenta el orden de aparición de las características, lo cual no se utiliza en otros estudios. Y así, describir distintas técnicas para mejorar el tiempo de cómputo al procesar el modelo.

2.5.12. Técnicas de optimización del tiempo de ejecución

Actualmente existen distintos métodos para analizar SPLs. Dependiendo de la información que sea necesario generar, unos serán mas eficientes que otros. Sin embargo, en el procesamiento de los modelos de variabilidad existe un problema latente, la explosión combinatoria al agregar al modelo nuevas características.

Para resolver u optimizar el problema de la explosión combinatoria existen distintos métodos. Entre ellos se propone traducir los modelos de variabilidad a problemas de factibilidad lógica [4]. También existen algoritmos para resolver problemas de factibilidad del coste óptimo, donde una vez superado el umbral de coste definido, no se seguirán procesando esos términos [20, 19, 3]. Otros estudios muestran el desarrollo de sistemas de tiempo real para modelar familias de productos, lo cual ha permitido que sean establecidos for-

malismos estándares empleando herramientas de modelado maduras como Uppaal Cora o Kronos, así como *frameworks* completos para modelar, simular y validar sistemas de tiempo real [1, 27, 18].

Es importante destacar que estos estudios se basan, o bien en resolver los problemas empleando técnicas y algoritmos de *branch and bound*, o empleando heurísticas para reducir el tiempo de procesamiento requerido para producir un resultado válido [56]. Por el contrario, en este trabajo de investigación se proponen y utilizan distintas técnicas que no sólo plantean heurísticas para reducir el tiempo total de procesamiento, sino que se permite describir relaciones de equivalencia, con respecto a otros términos del álgebra con estructuras más simples, que permitan generar el mismo conjunto de productos válidos.

2.5.13. Álgebras de procesos y análisis del coste

Actualmente existe un amplio rango de trabajos que relacionan el cálculo del coste en SPLs [82, 24, 58, 42, 55, 23, 94]. Generalmente estos trabajos calculan el coste de los productos como un todo, de forma que no consideran el orden en el que las características han sido producidas para generar un producto válido. Este aspecto es importante ya que el orden de procesamiento podría afectar al coste final del producto.

El coste calculado debe aportar información relevante en el modelo de variabilidad, información que será descrita, utilizando álgebras de procesos e interpretada, a medida que se estudia el comportamiento del sistema. Con el uso de las álgebras de procesos será posible mostrar la evolución y el comportamiento de propiedades estáticas en sistemas dinámicos [83, 64, 51, 29, 34, 49].

La relación entre la descripción formal de las SPLs empleando álgebras de procesos y el modelado del coste en el desarrollo de *software*, permite crear modelos formales que calculen el coste de los productos. Este coste puede tener en cuenta, o no, el orden en el que son procesadas sus características. Además, éste puede ser evaluado de manera dinámica a medida que se procesa el modelo de características [4, 30].

2.5.14. Probabilidades en álgebras de procesos

Las álgebras de procesos han mostrado ser una herramienta útil para describir el comportamiento de los sistemas. Dentro de la evolución de las mismas, se han extendido para cuantificar las transiciones entre los estados. Sin embargo, éstas han sido extendidas para soportar modelos probabilísticos [29, 71, 103, 13].

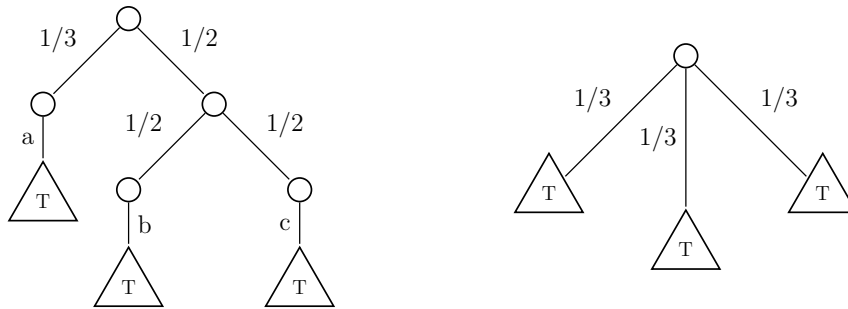


Figura 2.9: Sistemas de transiciones probabilísticos.

La figura 2.9 describe cómo se asignan probabilidades a las transiciones dentro del comportamiento del sistema. Es importante destacar que la probabilidad de una transición t no es necesariamente igual a las demás probabilidades del mismo nivel.

Estos modelos permiten asignar un número real, entre 0 y 1, a cada transición para que ocurra un determinado evento. Este caso es particularmente estudiado en esta tesis doctoral al producir las características o productos dentro de las SPLs.

Capítulo 3

SPLA: Marco teórico para representar diagramas FODA

“The job of formal methods is to elucidate the assumptions upon which formal correctness depends.”

Tony Hoare

Contenido

3.1. Sintaxis	28
3.2. Traducción de FODA a SPLA	31
3.3. Semántica Operacional	33
3.4. Semántica Denotacional	42
3.5. Reducción de los términos sintácticos	56
3.6. Semántica Axiomática	57
3.7. Consistencia en la traducción de FODA a SPLA	68
3.8. Factibilidad de los términos SPLA	70
3.9. Caso de estudio: Sistema VSS	83
3.10. Implementación de SPLA	90
3.10.1. Módulo de factibilidad	90
3.10.2. Módulo denotacional	90

Este capítulo presenta la definición e implementación del lenguaje formal **SPLA**, donde destacan los siguientes aspectos relevantes. En la sección 3.1 se define la sintaxis del lenguaje, la cual está compuesta por la descripción de los símbolos del álgebra y la definición de las reglas sintácticas para la construcción de términos **SPLA**. Seguidamente se presenta, en la sección 3.2, el proceso de traducción de diagramas **FODA** a **SPLA**. Describir este proceso correctamente es vital para el posterior análisis de **SPLA**, ya que en éste se muestra cómo

traducir un diagrama FODA a una estructura formal que puede ser procesada de manera automática. Las secciones 3.3 y 3.4 muestran la descripción formal de la semántica operacional y semántica denotacional, respectivamente. Estas semánticas permitirán dar sentido al proceso de cómputo sobre los términos del álgebra para generar productos válidos. La sección 3.5 muestra la demostración de que todo término del álgebra puede ser reescrito empleando una serie de elementos sintácticos elementales. La sección 3.6 demuestra que las igualdades deducidas de los axiomas del lenguaje son correctas, de igual manera, muestra la completitud del sistema ya que se describe como todas las identidades del modelo pueden ser deducidas del sistema de axiomas. La sección 3.7 presenta las definiciones y teoremas que permiten demostrar la consistencia de los términos traducidos de un diagrama FODA a un término SPLA. La sección 3.8 muestra la descripción del módulo de factibilidad, el cual permitirá determinar si existe algún producto que pertenezca al término P tal que cumpla todas las restricciones del modelo. La sección 3.9 muestra la descripción de un caso de estudio modelando un *software* para *streaming* de video. Este caso de estudio estará compuesto por la descripción del modelo de variabilidad, su traducción a SPLA y el procesamiento del modelo de acuerdo a las reglas de la semántica operacional y denotacional. La implementación del módulo de factibilidad y de la semántica denotacional serán descritos en la sección 3.10.

3.1. Sintaxis

La *sintaxis* está compuesta por los principios y reglas necesarias para construir *términos* propios del álgebra. En este trabajo se definirá una sintaxis que permita representar modelos de variabilidad utilizando FODA como modelo de referencia. Para definir la sintaxis, es necesario definir previamente el conjunto de *características* que formarán el término SPLA, denotado como \mathcal{F} , el cual será definido como el conjunto infinito de características. Además, A, B, C, \dots representarán características separadas, las cuales, a su vez, pertenecen a \mathcal{F} .

Una vez definidos los símbolos que describen a las características del álgebra, deben ser

definidos los símbolos que representarán a los operadores semánticos. En SPLA existen dos tipos de operadores. Por un lado son definidos los *operadores principales*, como $\cdot \vee \cdot$, $\cdot \wedge \cdot$, $A; \cdot$, $\bar{A}; \cdot$, $A \Rightarrow B \text{ in } \cdot$, $A \not\Rightarrow B \text{ in } \cdot$, los cuales tienen una correspondencia directa con los diagramas FODA y, por otro lado, existen *operadores auxiliares* tales como nil , \checkmark , $\cdot \setminus A$, $\cdot \Rightarrow A$, que son necesarios para definir la semántica del lenguaje.

Definición 3.1.1 Una *línea de productos software* es un término generado por la siguiente gramática:

$$P ::= \checkmark \mid \text{nil} \mid A; P \mid \bar{A}; P \mid \\ P \vee Q \mid P \wedge Q \mid A \not\Rightarrow B \text{ in } P \mid \\ A \Rightarrow B \text{ in } P \mid P \setminus A \mid P \Rightarrow A$$

donde $A, B \in \mathcal{F}$. Al conjunto de términos que pertenezcan al álgebra se les conocerá como SPLA. □

Con el fin de evitar el uso de paréntesis innecesarios, se asume que los operadores binarios son asociativos por la izquierda. Además, se asume la siguiente precedencia en los operadores (de mayor a menor prioridad): $A; P$, $\bar{A}; P$, $P \vee Q$, $P \wedge Q$, $A \not\Rightarrow B \text{ in } P$, $A \Rightarrow B \text{ in } P$, $A \Rightarrow B \text{ in } P$, $P \setminus A$, y $P \Rightarrow A$. De igual manera, para simplificar el procesamiento de los operadores de *selección única* ($\cdot \vee \cdot$) y de *conjunción ó paralelo* ($\cdot \wedge \cdot$), la proposición 3.3.8 enuncia que se mantienen las propiedades de conmutatividad y asociatividad en éstos, convirtiéndolos en operadores de aridad n en lugar de binarios.

La gramática descrita en la definición 3.1.1 describe que un término SPLA es una secuencia de operadores y características que representa, una vez procesado, el conjunto de *productos* válidos de la SPL. Básicamente, un producto es un conjunto de características que pueden ser producidas a partir de un término SPLA.

A continuación se procederá a describir el significado de cada operador. Además se ilustrará, con ejemplos, cómo deben ser procesados estos operadores empleando las reglas semánticas definidas en las secciones posteriores. En la sintaxis propuesta existen dos símbolos terminales, nil y \checkmark , los cuales son necesarios para definir la semántica del lenguaje. Estos símbolos serán utilizados al final del cómputo del término. En los casos donde pueda

procesarse el símbolo \checkmark , significará que ha sido generado un *producto válido*, por lo cual será necesario procesar inmediatamente el símbolo terminal `nil` para indicar que no debe ser procesado ningún otro elemento sintáctico en esa rama. Los operadores $A; P$ y $\bar{A}; P$ permiten agregar la característica A a cualquier producto obtenido de P , donde el operador $A; P$ indica que A es obligatorio, mientras que $\bar{A}; P$ indica que A es opcional. Por ejemplo, el término $A; \bar{B}; \checkmark$ representa una SPL con dos productos válidos. En este caso se obtiene un producto con la característica A y otro con las características A y B , ya que B es opcional.

Dentro de la definición inicial de FODA existen dos operadores binarios, $P \vee Q$ y $P \wedge Q$, donde el primero representa al operador de *selección única* y el segundo al operador de *conjunción*. Por ejemplo, el término $A; \checkmark \vee (B; \checkmark \vee C; \checkmark)$ generará tres productos válidos, cada uno con una característica, esto es, el primero con la característica A , el segundo con B y el tercero con C . Particularmente en este ejemplo, se muestran paréntesis agrupando los operadores de *selección única*, aunque es importante destacar que el operador \vee es conmutativo y asociativo, por lo que es posible reescribir el término anterior eliminando los paréntesis de la siguiente manera, $A; \checkmark \vee B; \checkmark \vee C; \checkmark$. De esta forma, el significado de este operador describe, básicamente, que debe ser seleccionada una característica entre n opciones. El término $A; (B; \checkmark \wedge C; \checkmark)$ describe un término válido del álgebra que incluye al operador de paralelo, esto es, un operador semántico que permitirá que las características a ambos lados del operador puedan ser procesadas donde, dependiendo si son procesadas las reglas de la semántica denotacional u operacional, se generará un sólo producto válido con todas las características ($\{ABC\}$) o productos combinando el orden en que las características son producidas ($\{[ABC], [ACB]\}$), respectivamente.

Las restricciones dentro de SPLA son modeladas de una manera sencilla e intuitiva. El operador $A \Rightarrow B \text{ in } P$ representa la restricción de *requerimiento*, mientras que el operador $A \not\Rightarrow B \text{ in } P$ muestra la restricción de *exclusion*. Por ejemplo, el término $A \Rightarrow B \text{ in } A; \checkmark$ generará un producto válido con las características A y B . De igual manera, si se considera el término $P = A; (B; \checkmark \vee C; \checkmark)$, serán generados dos productos válidos. El primero contendrá las

características A y B , mientras que el segundo contendrá las características A y C . Si al término anterior se agrega la siguiente restricción $A \not\Rightarrow B \text{ in } P$, entonces este nuevo término generará un único producto válido con las características A y C . El operador $P \Rightarrow A$ es necesario para definir el comportamiento del operador de requerimiento $A \Rightarrow B \text{ in } P$. Al generar los productos del término $A \Rightarrow B \text{ in } P$, se debe tener en cuenta si se ha procesado, o no, la característica A . En el caso de haber sido computada, se debe marcar el término para contemplar que la característica B debe ser procesada más adelante. El operador $P \Rightarrow B$ es utilizado con este propósito. La misma condición ocurre en el caso del operador $P \setminus B$. Al generarse los productos de $A \not\Rightarrow B \text{ in } P$, si A se procesa en algún punto, se debe marcar el término para que B no sea incluida. De esta manera el operador $P \setminus B$ indica que B está oculta.

3.2. Traducción de FODA a SPLA

La tabla 3.1 muestra la descripción gráfica del proceso utilizado para traducir los diagramas FODA a SPLA. De esta forma, utilizando las reglas descritas en esta tabla, cualquier diagrama FODA puede ser traducido o representado utilizando la sintaxis de SPLA. Este proceso de traducción de FODA a SPLA se lleva a cabo en tres pasos. Primero, se realiza la traducción del diagrama sin tener en cuenta las restricciones del modelo de variabilidad FODA (*requerimiento* y *exclusión*). Seguidamente se deben traducir las restricciones de *requerimiento*. El orden de aparición de las restricciones puede ser relevante, por lo que deben agregarse todas las restricciones en el mismo orden en que fueron procesadas en el diagrama. De esta forma, si la restricción de *requerimiento* $\boxed{A} \cdots \Rightarrow \boxed{B}$ y $\boxed{B} \cdots \Rightarrow \boxed{C}$ están en el diagrama, también debe ser agregada la restricción $\boxed{A} \cdots \Rightarrow \boxed{C}$. La proposición 3.7.1 demuestra que, si varias restricciones son transitivas, entonces el orden en el que éstas se aplican no es relevante. Finalmente, se procesan las restricciones de *exclusión*. En este caso, la proposición 3.7.3 demuestra que el orden de las restricciones de *exclusión* no es relevante.

Esta traducción se considera correcta por la proposición 3.7.1 y la proposición 3.7.3. Estas proposiciones no han sido incluidas en esta sección ya que es necesario presentar previamente


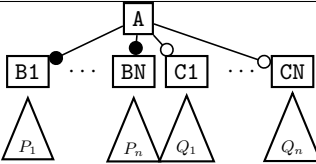
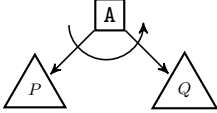
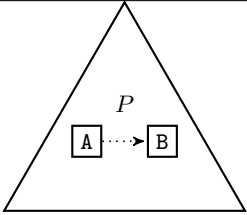
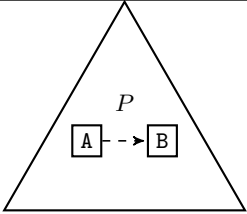
	Diagrama FODA	Término SPLA
Característica		$A; \checkmark$
Paralelo		$A; (B1; \triangle_{P_1} \wedge \dots \wedge BN; \triangle_{P_n} \wedge \overline{C1}; \triangle_{Q_1} \wedge \dots \wedge \overline{CN}; \triangle_{Q_n})$
Selección única		$A; (\triangle_P \vee \triangle_Q)$
Requerimiento		$A \Rightarrow B \text{ in } \triangle_P$
Exclusión		$A \not\Rightarrow B \text{ in } \triangle_P$

Tabla 3.1: Traducción de FODA a SPLA.

la definición de la relación de equivalencia, descrita en la sección 3.3 (ver definición 3.3.6).

Con el propósito de presentar de forma clara y sencilla los aspectos explicados en esta sección, la tabla 3.1 sólo considera diagramas de *selección única* con dos opciones. Sin embargo, se pueden representar diagramas n -arios con diagramas de *selección única*, ya que, tal y como se había comentado previamente, el operador \vee en SPLA es conmutativo y asociativo (ver proposición 3.3.8).

La figura 3.2 muestra ejemplos del proceso de traducción de diagramas FODA utilizando las reglas descritas en la tabla 3.1.

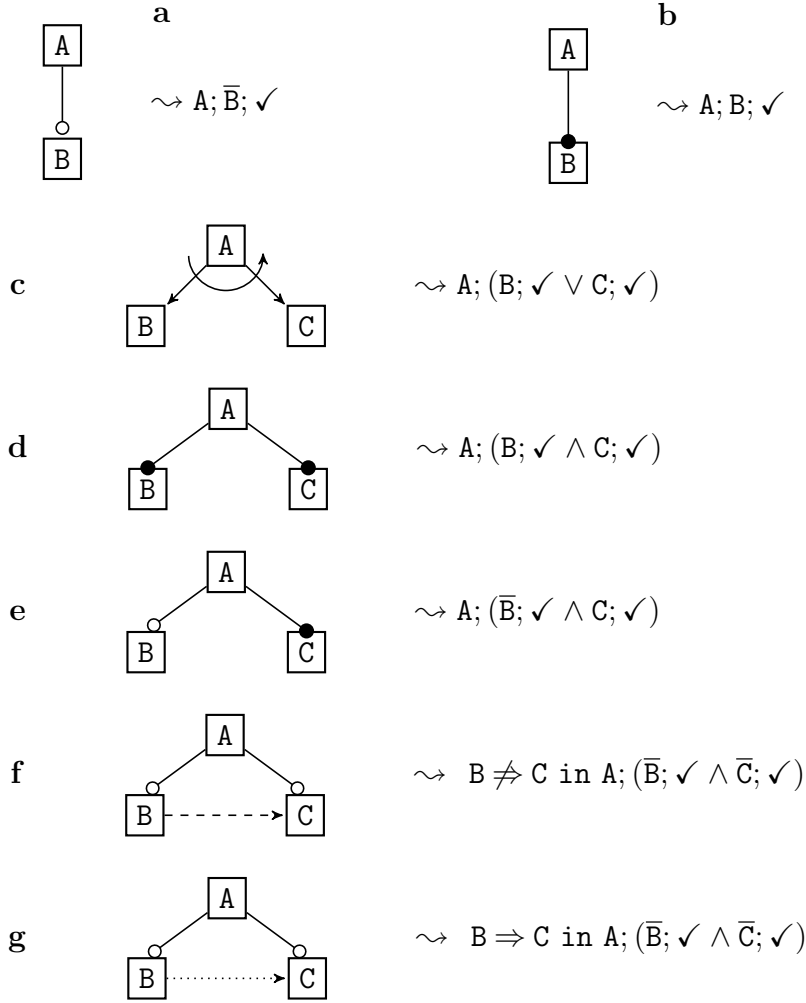


Figura 3.2: Ejemplos de traducciones de diagramas FODA a términos SPLA.

3.3. Semántica Operacional

En la sección anterior se ha definido la sintaxis del lenguaje para modelar SPLs utilizando SPLA. Sin embargo, es necesario definir las reglas semánticas que permitan representar el comportamiento del modelo. En esta sección se define el sistema de etiquetado de transiciones que permite representar cualquier término $P \in \text{SPLA}$. Las transiciones dentro del sistema de etiquetado de transiciones son *denotadas* por el conjunto $\mathcal{F} \cup \{\checkmark\}$, siendo \mathcal{F}

[tick]	$\checkmark \xrightarrow{\checkmark} \text{nil}$	[feat]	$A; P \xrightarrow{A} P$
[ofeat1]	$\bar{A}; P \xrightarrow{A} P$	[ofeat2]	$\bar{A}; P \xrightarrow{\checkmark} \text{nil}$
[cho1]	$\frac{P \xrightarrow{A} P_1}{P \vee Q \xrightarrow{A} P_1}$	[cho2]	$\frac{Q \xrightarrow{A} Q_1}{P \vee Q \xrightarrow{A} Q_1}$
[con1]	$\frac{P \xrightarrow{A} P_1}{P \wedge Q \xrightarrow{A} P_1 \wedge Q}$	[con2]	$\frac{Q \xrightarrow{A} Q_1}{P \wedge Q \xrightarrow{A} P \wedge Q_1}$
[con3]	$\frac{P \xrightarrow{\checkmark} \text{nil}, Q \xrightarrow{\checkmark} \text{nil}}{P \wedge Q \xrightarrow{\checkmark} \text{nil}}$	[req1]	$\frac{P \xrightarrow{C} P_1, C \neq A}{A \Rightarrow B \text{ in } P \xrightarrow{C} A \Rightarrow B \text{ in } P_1}$
[req2]	$\frac{P \xrightarrow{A} P_1}{A \Rightarrow B \text{ in } P \xrightarrow{A} P_1 \Rightarrow B}$	[req3]	$\frac{P \xrightarrow{\checkmark} \text{nil}}{A \Rightarrow B \text{ in } P \xrightarrow{\checkmark} \text{nil}}$
[excl1]	$\frac{P \xrightarrow{C} P_1, C \neq A \wedge C \neq B}{A \not\Rightarrow B \text{ in } P \xrightarrow{C} A \not\Rightarrow B \text{ in } P_1}$	[excl2]	$\frac{P \xrightarrow{A} P_1}{A \not\Rightarrow B \text{ in } P \xrightarrow{A} P_1 \setminus B}$
[excl3]	$\frac{P \xrightarrow{B} P_1}{A \not\Rightarrow B \text{ in } P \xrightarrow{B} P_1 \setminus A}$	[excl4]	$\frac{P \xrightarrow{\checkmark} \text{nil}}{A \not\Rightarrow B \text{ in } P \xrightarrow{\checkmark} \text{nil}}$
[forb1]	$\frac{P \xrightarrow{B} P_1, B \neq A}{P \setminus A \xrightarrow{B} P_1 \setminus A}$	[forb2]	$\frac{P \xrightarrow{\checkmark} \text{nil}}{P \setminus A \xrightarrow{\checkmark} \text{nil}}$
[mand1]	$\frac{P \xrightarrow{\checkmark} \text{nil}}{P \Rightarrow A \xrightarrow{A} \checkmark}$	[mand2]	$\frac{P \xrightarrow{A} P_1}{P \Rightarrow A \xrightarrow{A} P_1}$
[mand3]	$\frac{P \xrightarrow{B} P_1, A \neq B}{P \Rightarrow A \xrightarrow{B} P_1 \Rightarrow A}$		

$$A, B, C \in \mathcal{F}, a \in \mathcal{F} \cup \{\checkmark\}$$

Figura 3.3: Reglas de la semántica operacional de SPLA.

el conjunto de características y $\checkmark \notin \mathcal{F}$. En particular, si $A \in \mathcal{F}$, la transición $P \xrightarrow{A} Q$ significa que existe un producto de P que contiene la característica A . De igual manera, las transiciones de la forma $P \xrightarrow{\checkmark} \text{nil}$ muestran cuándo se ha producido un producto válido. Todas las reglas de la semántica operacional que permiten representar diagramas FODA como términos SPLA se presentan en la figura 3.3.

Definición 3.3.1 Dados los términos $P, Q \in \text{SPLA}$ y la característica $A \in \mathcal{F} \cup \{\checkmark\}$, existe una transición de P a Q etiquetada con el símbolo A , denotado por $P \xrightarrow{A} Q$, si puede

realizarse dicha transición a partir de las reglas descritas en la figura 3.3. \square

Antes de mostrar las propiedades de la semántica operacional, se deben justificar las reglas descritas en la figura 3.3, de la cual se obtendrán los productos de la SPL, calculando el conjunto de las trazas obtenidas al procesar todas las transiciones posibles de un término P .

El significado intuitivo de la regla **[tick]** expresa que se ha alcanzado un estado en el cual se ha generado un producto válido. Para lograr esto, es necesario el uso del operador auxiliar **nil**, el cual no generará ninguna otra transición. De esa forma, el procesamiento de las reglas se detendrá. Esto tiene como consecuencia que la única manera de generar un producto válido es cuando se procesa un término P , tal que $P \xrightarrow{\checkmark} \text{nil}$. Al ser **nil** un símbolo auxiliar, nunca deberá estar en un término inicial.

Las reglas **[feat]**, **[ofeat1]**, y **[ofeat2]** están relacionadas directamente con el cómputo de las características. La regla **[ofeat2]** muestra que se obtiene un producto válido sin tener en cuenta una característica opcional, lo cual significa que esta regla define la diferencia entre una característica obligatoria y una opcional. La característica A es opcional en P porque $P \xrightarrow{A} P_1$ y $P \xrightarrow{\checkmark} \text{nil}$. En este caso la transición $P \xrightarrow{\checkmark} \text{nil}$ muestra no sólo que P ha generado un producto válido, sino que también P puede procesar otras características y éstas son opcionales.

Las reglas **[cho1]** y **[cho2]** están relacionadas con el operador de *selección única*, de tal manera que al procesar $P \vee Q$, se debe escoger entre las características en P o las características en Q .

Las reglas **[con1]**, **[con2]** y **[con3]** representan al operador de *conjunción o paralelo*, cuyas reglas principales son **[con1]** y **[con2]**. Estas reglas muestran que el operador de paralelo es simétrico, lo cual indica que cualquier producto de $P \wedge Q$ debe contener las características de P y Q . La regla **[con3]** indica que para generar un producto válido, ambas partes del operador de paralelo deben poder procesar la regla **[tick]**.

Las reglas **[req1]**, **[req2]**, y **[req3]** están relacionadas con la restricción de *requerimiento*, lo que significa, de manera general, que una vez es procesada una característica, otra deba serlo de igual manera. La regla **[req1]** indica que $A \Rightarrow B \text{ in } P$ se comporta como P mientras que A no sea procesada. La regla **[req2]** indica que B es obligatoria una vez A es procesada. Finalmente, la regla **[req3]** es necesaria por el lema 3.3.2, lo cual indica que si las características de la restricción no fueron procesadas y P puede procesar **[tick]**, entonces será generado un producto válido.

Las reglas de exclusión, es decir **[excl1]**, **[excl2]**, **[excl3]** y **[excl4]**, modelan el comportamiento de la restricción de *exclusión* de una manera similar a las reglas de requerimiento. La regla **[excl1]** indica que $A \not\Rightarrow B \text{ in } P$ se comporta como P , siempre que P no produzca la característica A o la característica B . La regla **[excl2]** indica que una vez P haya producido la característica A , la característica B debe ser excluida u ocultada. La regla **[excl3]** describe el mismo comportamiento anterior, con la diferencia de que toma como referencia la otra característica de la restricción, lo que significa que cuando la característica B es computada, entonces la característica A debe ser excluida. La última regla de exclusión, **[excl4]**, describe de igual manera el comportamiento donde las características de la restricción no fueron procesadas. Así, P puede procesar **[tick]**, generando un producto válido.

Las reglas **[forb1]** y **[forb2]** hacen referencia a los operadores de ocultamiento introducidos por las reglas **[excl2]** y **[excl3]**, donde **[forb1]** establece que mientras que no sea procesada la característica oculta, el término podrá seguir siendo procesado, mientras que la regla **[forb2]** indica que una vez procesadas todas las características, si no se encuentra la característica oculta, entonces estará permitido generar un producto válido.

Las reglas **[mand1]**, **[mand2]** y **[mand3]** están relacionadas con el operador auxiliar $P \Rightarrow A$. La regla **[mand1]** indica que la característica A debe ser procesada antes de que sea generado un *producto válido*. La regla **[mand2]** hace referencia a que, una vez procesada la característica asociada a la relación $P \Rightarrow A$, entonces este operador debe ser eliminado del término restante. En el caso de **[mand3]** se indica que, si no se procesa la característica de

la relación $P \Rightarrow A$, entonces se mantendrá ésta dentro del término hasta que sea procesada. En caso contrario, si el término se procesa por completo, se ejecutará [mand1] para generar el producto válido.

A continuación se procederá a describir algunas propiedades de la semántica operacional.

Se consideran productos válidos los obtenidos al encontrar transiciones etiquetadas con el símbolo \checkmark . En este punto no pueden obtenerse, o procesarse, más características de la rama correspondiente, con lo que se establece el siguiente lema.

Lema 3.3.2 Dados los términos $P, Q \in \text{SPLA}$, si $P \xrightarrow{\checkmark} Q$ entonces $Q = \text{nil}$.

Demostración: La demostración es realizada por inducción al derivar $P \xrightarrow{\checkmark} Q$. Es necesario observar que, para todas las reglas que producen transiciones como $P \xrightarrow{\checkmark} Q$ se puede observar que Q es nil .

□

Una vez definida la semántica operacional del álgebra, es posible definir el concepto de las trazas de una SPL y, seguidamente, sus productos.

Definición 3.3.3 Una traza es una secuencia $s \in \mathcal{F}^*$, donde la traza vacía será denotada por ϵ . Dadas las trazas s_1 y s_2 , se define la concatenación de s_1 y s_2 como $s_1 \cdot s_2$. Dada la característica $A \in \mathcal{F}$ y la traza s , se dice que A pertenece a la traza s , escrito como $A \in s$, si y solo si existen trazas s_1 y s_2 tal que $s = s_1 \cdot A \cdot s_2$.

Además, es posible extender las transiciones descritas en la definición 3.3.1 a trazas. Dados los términos $P, Q, R \in \text{SPLA}$, se define de manera inductiva la transición $P \xrightarrow{s} R$ de la siguiente manera.

- $P \xrightarrow{\epsilon} P$.
- Si $P \xrightarrow{A} Q$ y $Q \xrightarrow{s} R$, entonces $P \xrightarrow{A \cdot s} R$.

□

Sólo las trazas terminadas con el símbolo \checkmark serán consideradas como productos válidos. De esta forma, para obtener los productos de una SPL, es necesario obtener todas trazas *exitosas* terminadas con el símbolo \checkmark . Es importante destacar que el símbolo \checkmark no es una característica, sino un símbolo auxiliar que describe la construcción de un producto válido y, por esa razón, no se incluye al calcular el conjunto de los productos válidos de la SPL.

Definición 3.3.4 Dado el término $P \in \text{SPLA}$ y $s \in \mathcal{F}^*$, s es una *traza exitosa* de P , escrita como $s \in \text{tr}(P)$, si y solo si $P \xrightarrow{s} Q \xrightarrow{\checkmark} \text{nil}$. \square

El orden en el que las características son producidas no se modela en FODA. Por esta razón, trazas distintas podrían representar el mismo producto. Por ejemplo, el producto obtenido de la traza **AB** es el mismo que el representado por la traza **BA**. De esta forma, para obtener los productos de una SPL se debe considerar el conjunto resultante de las trazas.

Definición 3.3.5 Dada la traza s , el *conjunto inducido por la traza* escrito como $[s]$, es el conjunto obtenido de los elementos de la traza sin considerar su posición dentro de ella.

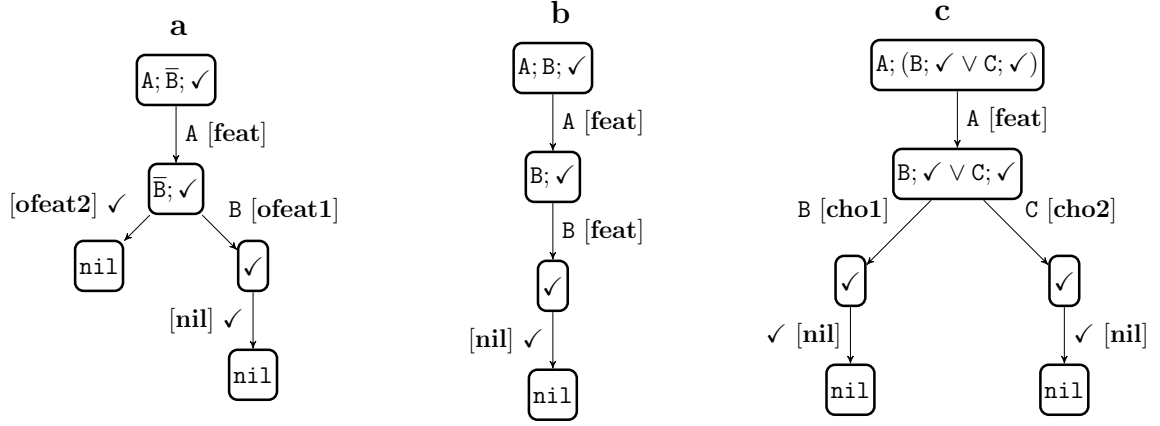
Dado un término $P \in \text{SPLA}$, se definen *los productos de P* como $\text{prod}(P)$, tal que, $\text{prod}(P) = \{[s] \mid s \in \text{tr}(P)\}$. \square

Seguidamente se muestra un ejemplo donde las posibles trazas del término $P = \mathbf{A}; (\bar{\mathbf{B}}; \checkmark \wedge \bar{\mathbf{C}}; \checkmark)$, son:

$$\begin{array}{lcl}
P \xrightarrow{\mathbf{A}} \bar{\mathbf{B}}; \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\checkmark} & \text{nil} \\
P \xrightarrow{\mathbf{A}} \bar{\mathbf{B}}; \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\mathbf{B}} \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\checkmark} \text{nil} \\
P \xrightarrow{\mathbf{A}} \bar{\mathbf{B}}; \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\mathbf{B}} \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\mathbf{C}} \checkmark \wedge \checkmark \xrightarrow{\checkmark} \text{nil} \\
P \xrightarrow{\mathbf{A}} \bar{\mathbf{B}}; \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\mathbf{C}} \bar{\mathbf{B}}; \checkmark \wedge \checkmark & \xrightarrow{\checkmark} \text{nil} \\
P \xrightarrow{\mathbf{A}} \bar{\mathbf{B}}; \checkmark \wedge \bar{\mathbf{C}}; \checkmark & \xrightarrow{\mathbf{C}} \bar{\mathbf{B}}; \checkmark \wedge \checkmark & \xrightarrow{\mathbf{B}} \checkmark \wedge \checkmark \xrightarrow{\checkmark} \text{nil}
\end{array}$$

Entonces, $\text{tr}(P) = \{\mathbf{A}, \mathbf{AB}, \mathbf{ABC}, \mathbf{AC}, \mathbf{ACB}\}$ y $\text{prod}(P) = \{[\mathbf{A}], [\mathbf{AB}], [\mathbf{ABC}], [\mathbf{AC}]\}$, ya que $[\mathbf{ABC}] = [\mathbf{ACB}]$.

Para ilustrar el procesamiento de las reglas de la semántica operacional, se utilizarán los ejemplos presentados en la figura 3.2.



$$\begin{aligned}
\mathbf{a} \quad & \text{prod}(A; \bar{B}; \checkmark) = \{ [A], [AB] \} \\
\mathbf{b} \quad & \text{prod}(A; B; \checkmark) = \{ [AB] \} \\
\mathbf{c} \quad & \text{prod}(A; (B; \checkmark \vee C; \checkmark)) = \{ [AB], [AC] \}
\end{aligned}$$

Figura 3.4: Aplicación de las reglas de la semántica operacional para los operadores $[feat]$, $[ofeat]$ y $[cho]$.

En los ejemplos **a** y **b** de la figura 3.4 se observa el comportamiento de las reglas que convierten una característica en opcional o hacen que sea obligatoria. En el ejemplo **a**, la característica B es opcional, mientras, que en **b** es obligatoria. Esta condición se muestra en el ejemplo **b**, puesto que existe un término correspondiente a la transición $\bar{B}; \checkmark \xrightarrow{\checkmark} nil$, el cual no existe en el ejemplo **a**.

Para el ejemplo **c** de la figura 3.4 y el ejemplo **d** de la figura 3.5 puede observarse la diferencia entre el operador de *selección única* y *conjunción*, respectivamente. En el operador de *selección única*, la parte no necesaria para procesar la siguiente característica *desaparece*, mientras que en el operador de *conjunción* se mantiene, ya que deberá procesarse en estados posteriores. Como resultado del procesamiento del término en el ejemplo **c**, se obtienen las trazas AB y AC , obteniendo dos productos distintos $[AB]$ y $[AC]$. Por el contrario, las trazas del ejemplo **d** son ABC y ACB , generando un único producto válido, $[ABC]$. En el ejemplo **e** se muestra el mismo término que en el ejemplo **d** con la única diferencia, de que

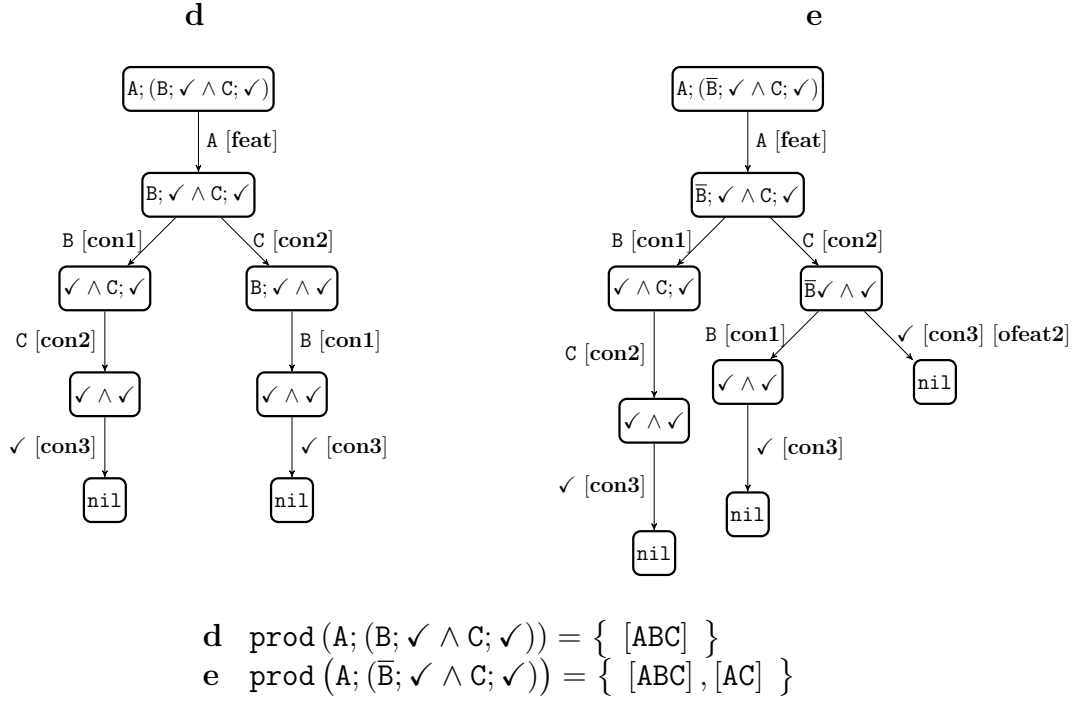


Figura 3.5: Aplicación de las reglas de la semántica operacional para el operador $[\text{con}]$.

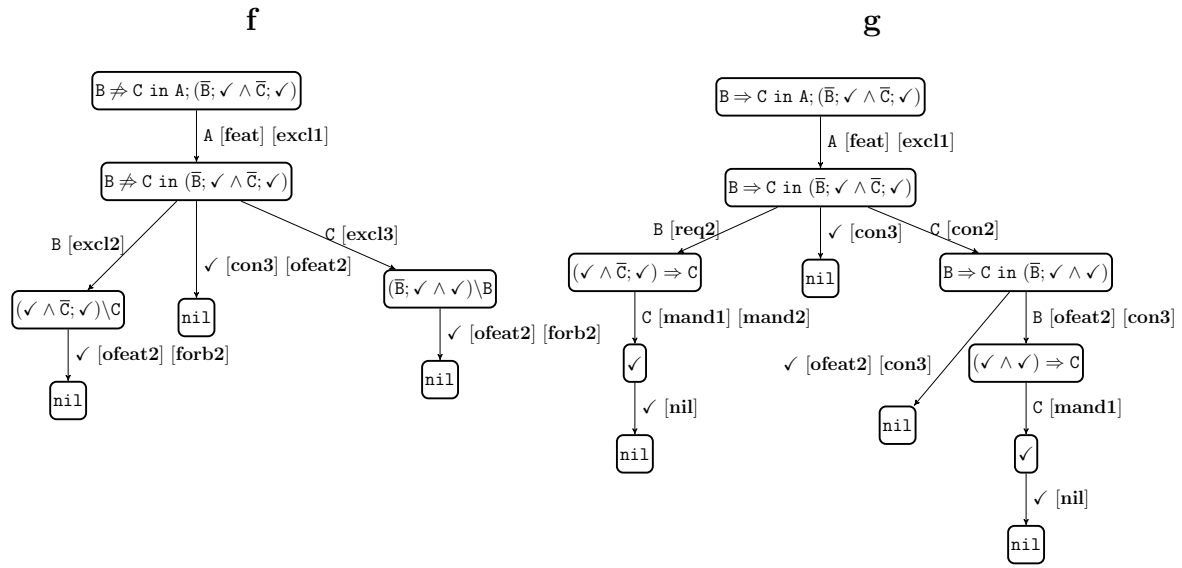
la característica B es opcional, generando los productos $[ABC]$ y $[AC]$.

La figura 3.6, en sus ejemplos **f** y **g**, describe el comportamiento de los operadores de *exclusión* y *requerimiento*, mostrando que una vez utilizadas estas reglas semánticas, serán procesados los operadores auxiliares de *ocultamiento* y *obligatoriedad*, respectivamente.

Una vez calculados los productos de una *SPL*, se define la relación de equivalencia entre ellos.

Definición 3.3.6 Dados los términos $P, Q \in \text{SPLA}$, se establece que P y Q son equivalentes, denotado por $P \equiv Q$, si los productos derivados de ambas *SPLs* son los mismos, esto es, $\text{prod}(P) = \text{prod}(Q)$. \square

Puesto que la relación \equiv está basada en la igualdad de los conjuntos, es también una relación de equivalencia. En la sección 3.4 se demuestra que esta afirmación es también una congruencia.



$$\mathbf{f} \text{ prod} \left(\frac{B \neq C \text{ in } A; (\bar{B}; \sqrt{\wedge} \bar{C}; \sqrt{)}}{(\sqrt{\wedge} \bar{C}; \sqrt{}) \setminus C} \right) = \{ [A], [AB], [AC] \}$$

$$\mathbf{g} \text{ prod} \left(\frac{B \Rightarrow C \text{ in } A; (\bar{B}; \sqrt{\wedge} \bar{C}; \sqrt{)}}{(\sqrt{\wedge} \bar{C}; \sqrt{}) \Rightarrow C} \right) = \{ [ABC], [A], [AC] \}$$

Figura 3.6: Aplicación de las reglas de la semántica operacional para los operadores $[\text{excl}]$ y $[\text{mand}]$.

Proposición 3.3.7 Dados los términos $P, Q, R \in \text{SPLA}$, las siguientes propiedades se mantienen:

- $P \equiv P$.
- Si $P \equiv Q$ entonces $Q \equiv P$.
- Si $P \equiv Q$ y $Q \equiv R$ entonces $P \equiv R$.

□

Seguidamente se muestran propiedades básicas del álgebra, tales como la conmutatividad y asociatividad de los operadores binarios. Estas propiedades son importantes ya que nos

permiten extender los operadores binarios a operadores n -arios

Proposición 3.3.8 Dados los términos $P, Q, R \in \text{SPLA}$, las siguientes propiedades se mantienen:

Conmutatividad $P \vee Q \equiv Q \vee P$ y $P \wedge Q \equiv Q \wedge P$.

Asociatividad $P \vee (Q \vee R) \equiv (P \vee Q) \vee R$ y $P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$.

□

3.4. Semántica Denotacional

La semántica *denotacional* es una representación más abstracta que la semántica operacional, ya que no depende de reglas que definen las transiciones entre un estado y otro. La descripción de esta semántica permitirá dar un significado a la construcción de modelos de variabilidad representados por FODA, a través de una serie de funciones semánticas descritas en secciones posteriores. Así se definirán las funciones que describen el comportamiento del modelo sin importar de la forma en que son procesadas.

En esta sección se describe la semántica denotacional para SPLA. Para poder definir esta semántica, es necesario establecer previamente el *dominio matemático* donde los elementos sintácticos de SPLA serán representados.

Como se ha descrito en la sección 3.3, la semántica de cualquier expresión SPLA está definida por su conjunto de productos, donde cada producto puede ser descrito por sus características. Una vez definido el dominio matemático necesario, $\mathcal{P}(\mathcal{P}(\mathcal{F}))^1$, siendo \mathcal{F} el conjunto de características, será necesario definir un operador semántico para cada operador sintáctico en SPLA. Esto se consigue mediante la siguiente definición.

Definición 3.4.1 Dados los conjuntos de productos $P, Q \in \mathcal{P}(\mathcal{P}(\mathcal{F}))$ y las características $A, B \in \mathcal{F}$, se definen los siguientes operadores:

¹Si X es un conjunto, $\mathcal{P}(X)$ describe el superconjunto de X .

- $\llbracket \text{nil} \rrbracket = \emptyset$

- $\llbracket \checkmark \rrbracket = \{\emptyset\}$

- $\llbracket \mathbf{A}; \cdot \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \mathbf{A}; \cdot \rrbracket(P) = \{\{\mathbf{A}\} \cup p \mid p \in P\}$$

- $\llbracket \overline{\mathbf{A}}; \cdot \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \overline{\mathbf{A}}; \cdot \rrbracket(P) = \{\emptyset\} \cup \{\{\mathbf{A}\} \cup p \mid p \in P\}$$

- $\llbracket \cdot \vee \cdot \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \times \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \cdot \vee \cdot \rrbracket(P, Q) = P \cup Q$$

- $\llbracket \cdot \wedge \cdot \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \times \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \cdot \wedge \cdot \rrbracket(P, Q) = \{p \cup q \mid p \in P, q \in Q\}$$

- $\llbracket \mathbf{A} \Rightarrow \mathbf{B} \text{ in } \cdot \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \mathbf{A} \Rightarrow \mathbf{B} \text{ in } \cdot \rrbracket(P) = \{p \mid p \in P, \mathbf{A} \notin p\} \cup \{p \cup \{\mathbf{B}\} \mid p \in P, \mathbf{A} \in p\}$$

- $\llbracket \mathbf{A} \nRightarrow \mathbf{B} \text{ in } \cdot \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \mathbf{A} \nRightarrow \mathbf{B} \text{ in } \cdot \rrbracket(P) = \{p \mid p \in P, \mathbf{A} \notin p\} \cup \{p \mid p \in P, \mathbf{B} \notin p\}$$

- $\llbracket \cdot \Rightarrow \mathbf{A} \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \cdot \Rightarrow \mathbf{A} \rrbracket(P) = \{p \cup \{\mathbf{A}\} \mid p \in P\}$$

- $\llbracket \cdot \setminus \mathbf{A} \rrbracket : \mathcal{P}(\mathcal{P}(\mathcal{F})) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{F}))$ como

$$\llbracket \cdot \setminus \mathbf{A} \rrbracket(P) = \{p \mid p \in P, \mathbf{A} \notin p\}$$

$ \begin{aligned} & \llbracket \checkmark \rrbracket = \{\emptyset\} \\ & \llbracket \bar{B}; \checkmark \rrbracket = \llbracket \bar{B}; \cdot \rrbracket(\llbracket \checkmark \rrbracket) = \llbracket \bar{B}; \cdot \rrbracket(\{\emptyset\}) = \{\emptyset, \{B\}\} \\ & \llbracket A; \bar{B}; \checkmark \rrbracket = \llbracket A; \cdot \rrbracket(\llbracket \bar{B}; \checkmark \rrbracket) = \llbracket A; \cdot \rrbracket(\{\emptyset\} \cup \{B\}) = \{\{A\}, \{A, B\}\} \end{aligned} $	$ \begin{aligned} & \llbracket \bar{B}; \checkmark \wedge C; \checkmark \rrbracket = \llbracket \cdot \wedge \cdot \rrbracket(\{\emptyset, \{B\}\}, \{\{C\}\}) = \{\{C\}, \{B, C\}\} \\ & \llbracket A; (\bar{B}; \checkmark \wedge C; \checkmark) \rrbracket = \llbracket A; \cdot \rrbracket(\llbracket \bar{B}; \checkmark \wedge C; \checkmark \rrbracket) = \llbracket A; \cdot \rrbracket(\{\{C\}, \{B, C\}\}) = \{\{A, C\}, \{A, B, C\}\} \end{aligned} $
$ \begin{aligned} & \llbracket B; \checkmark \rrbracket = \llbracket B; \cdot \rrbracket(\llbracket \checkmark \rrbracket) = \llbracket B; \cdot \rrbracket(\{\emptyset\}) = \{\{B\}\} \\ & \llbracket A; B; \checkmark \rrbracket = \llbracket A; \cdot \rrbracket(\llbracket B; \checkmark \rrbracket) = \llbracket A; \cdot \rrbracket(\{B\}) = \{\{A, B\}\} \end{aligned} $	$ \begin{aligned} & \llbracket \bar{B}; \checkmark \wedge \bar{C}; \checkmark \rrbracket = \llbracket \cdot \wedge \cdot \rrbracket(\llbracket \bar{B}; \checkmark \rrbracket, \llbracket \bar{C}; \checkmark \rrbracket) = \llbracket \cdot \wedge \cdot \rrbracket(\{\emptyset, \{B\}\}, \{\emptyset, \{C\}\}) = \{\emptyset, \{B\}, \{C\}, \{B, C\}\} \\ & \llbracket A; (\bar{B}; \checkmark \wedge \bar{C}; \checkmark) \rrbracket = \llbracket A; \cdot \rrbracket(\llbracket \bar{B}; \checkmark \wedge \bar{C}; \checkmark \rrbracket) = \llbracket A; \cdot \rrbracket(\{\emptyset, \{B\}, \{C\}, \{B, C\}\}) = \{\{A\}, \{A, B\}, \{A, C\}, \{A, B, C\}\} \end{aligned} $
$ \begin{aligned} & \llbracket B; \checkmark \rrbracket = \{\{B\}\} \\ & \llbracket C; \checkmark \rrbracket = \{\{C\}\} \\ & \llbracket B; \checkmark \vee C; \checkmark \rrbracket = \llbracket \cdot \vee \cdot \rrbracket(\llbracket B; \checkmark \rrbracket, \llbracket C; \checkmark \rrbracket) = \llbracket B; \checkmark \rrbracket \cup \llbracket C; \checkmark \rrbracket = \{\{B\}, \{C\}\} \\ & \llbracket A; (B; \checkmark \vee C; \checkmark) \rrbracket = \llbracket A; \cdot \rrbracket(\llbracket B; \checkmark \vee C; \checkmark \rrbracket) = \llbracket A; \cdot \rrbracket(\{\{B\}, \{C\}\}) = \{\{A, B\}, \{A, C\}\} \end{aligned} $	$ \begin{aligned} & \llbracket \left[\begin{array}{l} B \not\Rightarrow C \text{ in } A; \\ (\bar{B}; \checkmark \wedge \bar{C}; \checkmark) \end{array} \right] \rrbracket = \llbracket B \not\Rightarrow C \text{ in } \cdot \rrbracket \left(\begin{array}{l} \{\{A\}, \{A, B\}, \\ \{A, C\}, \{A, B, C\}\} \end{array} \right) = \{\{A\}, \{A, C\}, \{A, B\}\} \end{aligned} $
$ \begin{aligned} & \llbracket B; \checkmark \wedge C; \checkmark \rrbracket = \llbracket \cdot \wedge \cdot \rrbracket(\llbracket B; \checkmark \rrbracket, \llbracket C; \checkmark \rrbracket) = \llbracket \cdot \wedge \cdot \rrbracket(\{\{B\}\}, \{\{C\}\}) = \{\{B, C\}\} \\ & \llbracket A; (B; \checkmark \wedge C; \checkmark) \rrbracket = \llbracket A; \cdot \rrbracket(\llbracket B; \checkmark \wedge C; \checkmark \rrbracket) = \llbracket A; \cdot \rrbracket(\{\{B, C\}\}) = \{\{A, B, C\}\} \end{aligned} $	$ \begin{aligned} & \llbracket \left[\begin{array}{l} B \Rightarrow C \text{ in } A; \\ (\bar{B}; \checkmark \wedge \bar{C}; \checkmark) \end{array} \right] \rrbracket = \llbracket B \Rightarrow C \text{ in } \cdot \rrbracket \left(\begin{array}{l} \{\{A\}, \{A, B\}, \\ \{A, C\}, \{A, B, C\}\} \end{array} \right) = \{\{A\}, \{A, B, C\}, \{A, C\}\} \end{aligned} $

Figura 3.7: Aplicación de las reglas de la semántica denotacional.

□

Con estos operadores sobre conjuntos de productos se puede definir la semántica denotacional de cualquier expresión **SPLA**, la cual se define inductivamente de la manera habitual.

Definición 3.4.2 La semántica denotacional de **SPLA** se define inductivamente mediante la función $\llbracket \cdot \rrbracket : \text{SPLA} \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{F}))$, tal que, para cualquier operador n -ario² $\text{op} \in \{\text{nil}, \checkmark, A; \cdot, \bar{A}; \cdot, \cdot \vee \cdot, \cdot \wedge \cdot, A \Rightarrow B \text{ in } \cdot, A \not\Rightarrow B \text{ in } \cdot, \cdot \Rightarrow A, \cdot \setminus A\}$:

$$\llbracket \text{op}(P_1, \dots, P_n) \rrbracket = \llbracket \text{op} \rrbracket(\llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket)$$

□

Con el fin de ilustrar la semántica denotacional descrita, se ha aplicado ésta a los ejemplos presentados en la figura 3.2. Los resultados obtenidos se presentan en la figura 3.7.

² nil y \checkmark son operadores 0-arios. $A; \cdot$, $\bar{A}; \cdot$, $A \Rightarrow B \text{ in } \cdot$, $A \not\Rightarrow B \text{ in } \cdot$, $\cdot \Rightarrow A$, $\cdot \setminus A$ son operadores 1-arios. $\cdot \vee \cdot$ y $\cdot \wedge \cdot$ son operadores binarios.

El resto de esta sección está dedicada a comprobar que el conjunto de los productos generados por la semántica operacional coincide con los productos generados por la semántica denotacional. Para ello, se definen los resultados auxiliares que relacionan la semántica operacional con los operadores denotacionales de la definición 3.4.1.

El primer resultado está relacionado a la finalización de la construcción de la traza, donde los productos de una SPL se generan desde las hojas del árbol sintáctico, hasta la raíz. Esto significa que el primer producto *procesado* por la semántica denotacional es el producto sin características, donde se muestra que $\{\emptyset\} = \text{prod}(\checkmark) = \llbracket \checkmark \rrbracket$. Sin embargo, $\emptyset = \text{prod}(\text{nil}) = \llbracket \text{nil} \rrbracket$.

Lema 3.4.3 Dado el término $P \in \text{SPLA}$, si $P \xrightarrow{\checkmark} \text{nil}$, entonces $\emptyset \in \llbracket P \rrbracket$.

Demostración: Se demuestra por inducción sobre la longitud de la transición $P \xrightarrow{\checkmark} \text{nil}$. Consideremos que tiene longitud n .

- **$n = 0$:** Existen dos casos: $P = \checkmark$ y $P = \bar{A}; P$, para ambos casos $\emptyset \in \llbracket P \rrbracket$.
- **$n > 1$:** En este caso, se deben analizar las reglas que producen $P \xrightarrow{\checkmark} \text{nil}$. Estas reglas son **[cho1]**, **[cho2]**, **[con3]**, **[req3]**, **[excl4]** y **[forb2]**.
 - **Reglas [cho1] y [cho2]:** Estas reglas son simétricas, de tal manera que centraremos la demostración en una de ellas, por ejemplo **[cho1]**. En este caso $P = P_1 \vee P_2$ y $P_1 \xrightarrow{\checkmark} \text{nil}$. Por inducción, $\emptyset \in \llbracket P_1 \rrbracket$ utilizando la definición 3.4.1, se obtiene $\emptyset \in \llbracket P \rrbracket$.
 - **Regla [con3]:** En este caso $P = P_1 \wedge P_2$, $P_1 \xrightarrow{\checkmark} \text{nil}$ y $P_2 \xrightarrow{\checkmark} \text{nil}$. Por inducción $\emptyset \in \llbracket P_1 \rrbracket$ y $\emptyset \in \llbracket P_2 \rrbracket$. Se obtiene el resultado por medio de la definición 3.4.1.
 - **Regla [req3]:** En este caso $P = A \Rightarrow B \text{ in } P_1$ y $P_1 \xrightarrow{\checkmark} \text{nil}$. Por inducción $\emptyset \in \llbracket P_1 \rrbracket$. Desde que $A \notin \emptyset$ utilizando la definición 3.4.1, se obtiene $\emptyset \in \llbracket P \rrbracket$.
 - **Regla [excl4]:** En este caso $P = A \not\Rightarrow B \text{ in } P_1$ y $P_1 \xrightarrow{\checkmark} \text{nil}$. Por inducción $\emptyset \in \llbracket P_1 \rrbracket$. Desde que $A \notin \emptyset$ y $B \notin \emptyset$ utilizando la definición 3.4.1, se obtiene $\emptyset \in \llbracket P \rrbracket$.

- **Regla [req3]:** En este caso $P = P_1 \setminus A$ y $P_1 \xrightarrow{\checkmark} \text{nil}$. Por inducción $\emptyset \in \llbracket P_1 \rrbracket$. Desde que $A \notin \emptyset$ utilizando la definición 3.4.1, se obtiene $\emptyset \in \llbracket P \rrbracket$.

□

A continuación se presenta un lema para cada operador de la sintaxis. Cada lema indica que su correspondiente operador semántico está definido en la definición 3.4.1. Estos resultados son necesarios para el caso inductivo del teorema 3.4.12.

Proposición 3.4.4 Dado el término $P \in \text{SPLA}$ y la característica $A \in \mathcal{F}$, entonces

$$\text{prod}(A; P) = \llbracket A; \rrbracket(\text{prod}(P)).$$

Demostración: Dado que la única regla de la semántica operacional que puede ser aplicada al término $A; P$ es [feat], se obtiene $\text{tr}(A; P) = \{A \cdot s \mid s \in \text{tr}(P)\}$. Así, $\text{prod}(A; P) = \{A \cup p \mid p \in \text{prod}(P)\}$, que se corresponde con la definición de $\llbracket A; \rrbracket(\text{prod}(P))$ (ver definición 3.4.1). □

Proposición 3.4.5 Dado el término $P \in \text{SPLA}$ y la característica $A \in \mathcal{F}$, entonces

$$\text{prod}(\bar{A}; P) = \llbracket \bar{A}; \rrbracket(\text{prod}(P)).$$

Demostración: Las reglas de la semántica operacional que pueden ser aplicadas a $\bar{A}; P$ son [ofeat1] y [ofeat2]. Con lo que se obtiene $\text{tr}(\bar{A}; P) = \{\checkmark\} \cup \{A \cdot s \mid s \in \text{tr}(P)\}$. Por lo tanto, $\text{prod}(\bar{A}; P) = \{\emptyset\} \cup \{A \cup p \mid p \in \text{prod}(P)\}$, que corresponde con la definición de $\llbracket \bar{A}; \rrbracket(\text{prod}(P))$ (ver definición 3.4.1). □

Proposición 3.4.6 Dados los términos $P, P' \in \text{SPLA}$, entonces

$$\text{prod}(P \vee P') = \llbracket \vee \rrbracket(\text{prod}(P), \text{prod}(P')).$$

Demostración: De las reglas de la semántica operacional [cho1] y [cho2] se obtiene $\text{tr}(P \vee P') = \text{tr}(P) \cup \text{tr}(P')$. De tal manera que

$$\text{prod}(P \vee P') = \text{prod}(P) \cup \text{prod}(P') = \llbracket \vee \rrbracket(\text{prod}(P), \text{prod}(P'))$$

□

Proposición 3.4.7 Dados los términos $P, P' \in \text{SPLA}$ y las características $A, B \in \mathcal{F}$, entonces $\text{prod}(P \wedge P') = \llbracket \wedge \rrbracket(\text{prod}(P), \text{prod}(P'))$.

Demostración: Primero, considérese $[s] \in \text{prod}(P \wedge P')$, será demostrado que la traza $[s] \in \llbracket \wedge \rrbracket(\text{prod}(P), \text{prod}(P'))$ por inducción de la longitud de s .

- $s = \epsilon$: En este caso, $[s] = \emptyset$ y $P \wedge P' \xrightarrow{\checkmark} \text{nil}$. Solo es posible aplicar la regla de la semántica operacional **[con3]** para obtener esta transición. Entonces $P \xrightarrow{\checkmark} \text{nil}$ y $P' \xrightarrow{\checkmark} \text{nil}$. Así $\emptyset \in \text{prod}(P)$ y $\emptyset \in \text{prod}(P')$. Entonces, se cumple que $[s] = \emptyset \in \llbracket \wedge \rrbracket(\text{prod}(P), \text{prod}(P'))$ por la definición 3.4.1.
- $s = A \cdot s'$: En este caso, $P \wedge P' \xrightarrow{A} P''$. Es posible aplicar las reglas de la semántica operacional **[con1]** o **[con2]** para obtener esta transición. La regla **[con2]** es simétrica con respecto a la regla **[con1]**, entonces es posible concentrarse en **[con1]**. P_1 existe, así que $P \xrightarrow{A} P_1$, $P'' = P_1 \wedge P'$ y $[s'] \in \text{prod}(P_1 \wedge P')$. Por inducción se obtiene que $[s'] \in \llbracket \wedge \rrbracket(\text{prod}(P_1), \text{prod}(P'))$, así que existen productos $p_1 \in \text{prod}(P_1)$ y $p_2 \in \text{prod}(P')$ tal que $[s'] = p_1 \cup p_2$. Considérese la traza $s_1 \in \text{tr}(P_1)$ de tal manera que $[s_1] = p_1$. Entonces

$$[s] = \{A\} \cup [s'] = \{A\} \cup p_1 \cup p_2 = (\{A\} \cup [s_1]) \cup p_2$$

Finalmente es obtenido el resultado por la definición 3.4.1 desde que $A \cdot s_1 \in \text{tr}(P)$ y $p_2 \in \text{prod}(P')$.

Ahora considérese $p \in \llbracket \wedge \rrbracket(\text{prod}(P), \text{prod}(P'))$. Utilizando la definición 3.4.1 existen $p_1 \in \text{prod}(P)$ y $p_2 \in \text{prod}(P')$ tal que $p = p_1 \cup p_2$. Entonces, existen las trazas satisfactorias $s_1 \in \text{tr}(P)$ y $s_2 \in \text{tr}(P')$ tal que $[s_1] = p_1$ y $[s_2] = p_2$. La demostración es realizada por inducción sobre la suma de las longitudes de s_1 y s_2 .

- $\text{len}(s_1) + \text{len}(s_2) = 0$: Entonces, $s_1 = \epsilon$, $s_2 = \epsilon$ y $p = \emptyset$. En este caso se tienen las transiciones $P \xrightarrow{\checkmark} \text{nil}$ y $P' \xrightarrow{\checkmark} \text{nil}$. Al aplicar la regla **[con5]**, se obtiene la transición $P \wedge P' \xrightarrow{\checkmark} \text{nil}$. Por lo tanto $\emptyset \in \text{prod}(P \wedge P')$.

- **$len(s_1) + len(s_2) > 0$:** Suponemos que $s_1 = A \cdot s'_1$, (el caso $s_2 = A \cdot s_2$ es simétrico). Entonces, existe una transición $P \xrightarrow{A} P_1$ tal que s'_1 es una traza satisfactoria de P_1 . Utilizando la definición 3.4.1, $[s'_1] \cup [s_2] \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P_1), \text{prod}(P'))$. Por inducción se obtiene que $[s'_1] \cup [s_2] \in \text{prod}(P_1 \wedge P')$. Al aplicar la regla **[con1]**, obtenemos la transición $P \wedge P' \xrightarrow{A} P_1 \wedge P'$. Entonces $\{A\} \cup [s'_1] \cup [s_2] \in \text{prod}(P \wedge P')$. De esta manera se obtiene el resultado deseado ya que $[s_1] = \{A\} \cup [s'_1]$.

□

Proposición 3.4.8 Dados los términos $P, P' \in \text{SPLA}$ y las características $A, B \in \mathcal{F}$, entonces $\text{prod}(P \Rightarrow A) = \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P))$.

Demostración:

Primero será demostrado

$$\text{prod}(P \Rightarrow A) \subseteq \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P))$$

Entonces, considérese $p \in \text{prod}(P \Rightarrow A)$. Una traza satisfactoria $s \in \text{tr}(P \Rightarrow A)$, existe de tal manera que $[s] = p$. Se demostrará que $p \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P))$ por inducción sobre la longitud de s . Por la regla de la semántica operacional **[mand1]**, $s \neq \epsilon$. Así que el caso base se da cuando $s = A$ ($p = \{A\}$). También, por la regla **[mand1]** se obtiene la transición $P \xrightarrow{\checkmark} \text{nil}$, entonces $\emptyset \in \text{prod}(P)$. Por lo tanto, por la definición 3.4.1 $p = \{A\} \cup \emptyset \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P))$.

Considérese $s = B \cdot s'$. Si $A = B$, se obtiene la transición $P \Rightarrow A \xrightarrow{A} P_1$ y $s' \in \text{tr}(P_1)$. Esta transición solo puede ser deducida mediante la regla **[mand2]**. Por lo tanto, $P \xrightarrow{A} P_1$ y $s \in \text{tr}(P)$. Para obtener el resultado solo es necesario tomar en cuenta que la característica $A \in [s]$, entonces $[s] = \{A\} \cup [s']$. Si $A \neq B$, se obtiene la transición $P \Rightarrow A \xrightarrow{B} P_1 \Rightarrow A$ y $s' \in \text{tr}(P_1 \Rightarrow A)$. Esta transición, solo puede ser deducida utilizando la regla **[mand3]**. Por lo tanto, $P \xrightarrow{B} P_1$. Por inducción, se obtiene que $[s'] \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P_1))$. Así, mediante la definición 3.4.1, existe un $p \in \text{prod}(P_1)$, tal que $p \cup \{A\} = [s']$. También, debe existir

$s_1 \in \text{tr}(P_1)$, tal que $[s_1] = p_1$. De esta manera puede observarse que $B \cdot s_1 \in \text{tr}(P)$, así $\{B\} \cup p_1 \in \text{prod}(P)$. Entonces,

$$p = [s] = \{B\} \cup [s'] = \{B\} \cup p_1 \cup \{A\} \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P))$$

Ahora se debe demostrar que

$$\llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P)) \subseteq \text{prod}(P \Rightarrow A)$$

Considérese, que $p \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P))$. Entonces, utilizando la definición 3.4.1, existe un producto $p' \in \text{prod}(P)$, tal que $p = p' \cup \{A\}$. También, existe una traza $s \in \text{tr}(P)$, tal que $[s] = p'$. Será demostrado que el producto $p \in \text{prod}(P \Rightarrow A)$ por inducción de la longitud de s .

- $s = \epsilon$: En este caso se tiene que $P \xrightarrow{\checkmark} \text{nil}$ y $p' = \emptyset$. Entonces, al aplicar las reglas de la semántica operacional **[mand1]** y **[tick]**, se obtiene

$$P \Rightarrow A \xrightarrow{A} \checkmark \xrightarrow{\checkmark} \text{nil}$$

De esta manera, $p = p' \cup \{A\} = \{A\} \in \text{prod}(P \Rightarrow A)$.

- $s = A \cdot s'$: En este caso se tiene la transición $P \xrightarrow{A} P'$ y la traza $s' \in \text{tr}(P')$. Entonces, mediante la regla de la semántica operacional **[mand2]**, se obtiene que $P \Rightarrow A \xrightarrow{A} P'$. Por lo tanto, $s \in \text{tr}(P \Rightarrow A)$. Para obtener el resultado, es suficiente tomar en cuenta que $A \in [s]$, entonces $p = p' \cup \{A\} = [s] \cup \{A\} = [s]$.
- $s = B \cdot s' \text{ con } A \neq B$: En este caso se tiene $P \xrightarrow{B} P'$ y $s' \in \text{tr}(P')$. Por medio de la definición 3.4.1, se obtiene

$$\{A\} \cup [s'] \in \llbracket \cdot \Rightarrow A \rrbracket(\text{prod}(P'))$$

Y por inducción, se obtiene

$$\{A\} \cup [s'] \in \text{prod}(P' \Rightarrow A)$$

Existe una traza $s'' \in \text{tr}(P' \Rightarrow A)$, tal que $[s''] = \{A\} \cup [s']$. Al aplicar la regla **[mand3]**, se obtiene que $P \Rightarrow A \xrightarrow{B} P' \Rightarrow A$. Finalmente, ya que $B \cdot s'' \in \text{tr}(P \Rightarrow A)$, se obtiene

$$p = p' \cup \{A\} = [s] \cup \{A\} = \{B\} \cup [s'] \cup \{A\} = \{B\} \cup [s''] \in \text{prod}(P \Rightarrow A)$$

□

Proposición 3.4.9 Dados los términos $P, P' \in \text{SPLA}$ y las características $A, B \in \mathcal{F}$, entonces $\text{prod}(P \setminus A) = \llbracket \cdot \setminus A \rrbracket(\text{prod}(P))$.

Demostración: Primero es necesario demostrar que

$$\text{prod}(P \setminus A) \subseteq \llbracket \cdot \setminus A \rrbracket(\text{prod}(P))$$

Considérese el producto $p \in \text{prod}(P \setminus A)$. Una traza satisfactoria $s \in \text{tr}(P \setminus A)$, existe de tal manera que $[s] = p$. Se mostrará que $p \in \llbracket \cdot \setminus A \rrbracket(\text{prod}(P))$ por inducción sobre la longitud de s .

- $s = \epsilon$: En este caso, se tiene que $p = \emptyset$ y $P \setminus A \xrightarrow{\checkmark} \text{nil}$. La única regla que puede aplicarse es la regla **[forb2]**, así que, $P \xrightarrow{\checkmark} \text{nil}$. Empleando el lema 3.4.3, entonces $\emptyset \in \text{prod}(P)$. Utilizando la definición 3.4.1 se obtiene que $p = \emptyset \in \text{prod}(P \setminus A)$.
- $s = B \cdot s'$: En este caso, $P \setminus A \xrightarrow{B} P' \setminus A$ y $s' \in \text{tr}(P' \setminus A)$. Ya que la traza s es satisfactoria, s' también es satisfactoria. Así que $[s'] \in \text{prod}(P' \setminus A)$. Por inducción se obtiene que $[s'] \in \llbracket \cdot \setminus A \rrbracket(P')$, por lo tanto $[s'] \in \text{prod}(P')$ y $A \notin [s']$. La regla **[forb1]**, es la única regla de la semántica operacional que puede utilizarse para deducir $P \setminus A \xrightarrow{B} P' \setminus A$, para así obtener $B \cdot s' \in \text{tr}(P)$ y $B \neq A$. Por lo tanto, $[s] \in \text{tr}(P)$ y $A \notin [s]$, entonces utilizando la definición 3.4.1, se obtiene que $p = [s] \in \llbracket \cdot \setminus A \rrbracket(\text{prod}(P))$.

Ahora se debe demostrar

$$\llbracket \cdot \setminus A \rrbracket(\text{prod}(P)) \subseteq \text{prod}(P \setminus A)$$

Considérese el producto $p \in \llbracket \cdot \backslash A \rrbracket(\text{prod}(P))$. Utilizando la definición 3.4.1, $p \in \text{prod}(P)$ y $A \notin p$. Por lo tanto, existe una traza satisfactoria $s \in \text{tr}(P)$, tal que $[s] = p$. Desde que $A \notin [s]$, se obtiene que s es una traza satisfactoria de $P \backslash A$. Por lo tanto, $p = [s] \in \text{prod}(P \backslash A)$.

□

Proposición 3.4.10 Dados los términos $P, P' \in \text{SPLA}$ y las características $A, B, C \in \mathcal{F}$, entonces $\text{prod}(A \Rightarrow B \text{ in } P) = \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$.

Demostración: Primero, se debe demostrar que

$$\text{prod}(A \Rightarrow B \text{ in } P) \subseteq \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$$

Entonces, considérese el producto $p \in \text{prod}(A \Rightarrow B \text{ in } P)$. Existe una traza satisfactoria $s \in \text{tr}(A \Rightarrow B \text{ in } P)$, tal que $[s] = p$. Sera demostrado, que $p \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$ por inducción sobre la longitud de s .

- $s = \epsilon$: En este caso, se tiene la transición

$$\text{prod}(A \Rightarrow B \text{ in } P) \xrightarrow{\checkmark} \text{nil}$$

La única regla de la semántica operacional que puede aplicarse, es la regla [req3]. Entonces, $P \xrightarrow{\checkmark} \text{nil}$, y por lo tanto $\emptyset \in \text{prod}(P)$. De esta manera y utilizando definición 3.4.1, se obtiene que $\emptyset \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$.

- $s = C \cdot s'$ con $C \neq A$: En este caso, se tiene la transición $A \Rightarrow B \text{ in } P \xrightarrow{C} P'$. Si $A \neq C$, utilizando la regla [req1], se obtiene que $P' = A \Rightarrow B \text{ in } P_1$ con $P \xrightarrow{C} P_1$ y $s' \in \text{tr}(A \Rightarrow B \text{ in } P_1)$. Por inducción, se obtiene que $[s'] \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P_1))$.

Mediante el uso de la definición 3.4.1, se obtienen dos casos:

- $A \notin [s']$: En este caso $[s'] \in \text{prod}(P_1)$, por lo tanto $\{C\} \cup [s'] \in \text{prod}(P)$, y desde que $C \neq A$ utilizando la definición 3.4.1, se obtiene que $\{C\} \cup [s'] \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$

- $A \in [s']$: En este caso, utilizando la definición 3.4.1, existe una traza $s'' \in \text{tr}(P_1)$, tal que $[s'] = [s''] \cup \{B\}$. Por medio de la definición 3.4.1, se obtiene que $\{C\} \cup [s'] \cup \{B\} \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$.
- $s = A \cdot s'$: En este caso, se tiene la transición

$$A \Rightarrow B \text{ in } P \xrightarrow{A} P'$$

Desde que la única regla de la semántica operacional que es aplicable es la regla **[req2]**, se obtiene que existe un término $P_1 \in \text{SPLA}$, tal que $P \xrightarrow{A} P_1$, $P' = P_1 \Rightarrow B$ y $s' \in \text{tr}(P_1 \Rightarrow B)$. Utilizando el lema 3.4.8, se obtiene que $[s'] \in \llbracket \cdot \Rightarrow B \rrbracket \text{prod}(P_1)$. Empleando la definición 3.4.1, existe una traza $s'' \in \text{tr}(P_1)$, tal que $[s'] = [s''] \cup \{B\}$, ya que $P \xrightarrow{A} P_1$ y $A \cdot s'' \in \text{tr}(P)$. Por lo tanto, empleando la definición 3.4.1, se obtiene que

$$p = [s] = \{A\} \cup [s'] = \{A\} \cup [s''] \cup \{B\} \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$$

Ahora se debe demostrar que

$$\llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P)) \subseteq \text{prod}(A \Rightarrow B \text{ in } P)$$

Considerando $p \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$. Empleando la definición 3.4.1 existen dos casos:

- $A \notin p$: En este caso el producto $p \in \text{prod}(P)$, tal que existe una traza $s \in \text{tr}(P)$, y de esta manera $[s] = p$. Desde que $A \notin P$, al aplicar la regla de la semántica operacional **[req1]**, se obtiene que $s \in \text{tr}(A \Rightarrow B \text{ in } P)$.
- **Existe un producto $p' \in \text{prod}(P)$, tal que $A \in p'$ y $p = p' \cup \{B\}$** : Considerando la traza $s \in \text{tr}(P)$ tal que $p' = [s]$. Entonces, existen las trazas s_1 y s_2 , tal que $s = s_1 \cdot A \cdot s_2$ y $A \notin s_1$. Y, existen los términos P_1 y P_2 , tal que $P \xRightarrow{s_1} P_1 \xrightarrow{A} P_2$ y $s_2 \in \text{tr}(P_2)$. De esta manera, al aplicar la regla **[req1]** se obtiene,

$$A \Rightarrow B \text{ in } P \xRightarrow{s_1} A \Rightarrow B \text{ in } P_1 \xrightarrow{A} P_2 \Rightarrow B$$

Desde que $[s_2] \in \text{prod}(P_2)$, utilizando el lema 3.4.8 se obtiene que $[s_2] \cup \{B\} \in \text{prod}(P_2 \Rightarrow B)$. Considérese $s'_2 \in \text{tr}(P_2 \Rightarrow B)$, tal que $[s'_2] = [s_2] \cup \{B\}$. Por lo tanto, $s_1 \cdot A \cdot s'_2 \in \text{tr}(A \Rightarrow B \text{ in } P)$, así que se obtiene

$$p = p' \cup \{B\} = [s_1] \cup \{A\} \cup [s_2] \cup \{B\} = [s_1] \cup \{A\} \cup [s'_2] \in \text{prod}(A \Rightarrow B \text{ in } P)$$

□

Proposición 3.4.11 Dados los términos $P, P' \in \text{SPLA}$ y las características $A, B, C \in \mathcal{F}$, entonces $\text{prod}(A \not\Rightarrow B \text{ in } P) = \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$.

Demostración: Primero, es necesario demostrar que

$$\text{prod}(A \not\Rightarrow B \text{ in } P) \subseteq \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$$

Entonces, considérese el término $p \in \text{prod}(A \not\Rightarrow B \text{ in } P)$. Existe una traza satisfactoria $s \in \text{tr}(A \not\Rightarrow B \text{ in } P)$, tal que $[s] = p$. Será demostrado, que $p \in \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$ por inducción sobre la longitud de s .

- $s = \epsilon$: En este caso, se tiene la transición

$$\text{prod}(A \not\Rightarrow B \text{ in } P) \xrightarrow{\checkmark} \text{nil}$$

La única regla de la semántica operacional que puede aplicarse es la regla [excl4]. Así, que $P \xrightarrow{\checkmark} \text{nil}$ y por lo tanto $\emptyset \in \text{prod}(P)$. Entonces, utilizando la definición 3.4.1, se tiene que $\emptyset \in \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$.

- $s = C \cdot s'$ con $C \neq A$ y $C \neq B$: En este caso, se tiene la transición $A \not\Rightarrow B \text{ in } P \xrightarrow{C} P'$. Si $A \neq C$, al aplicar la regla [excl1], se obtiene que $P' = A \not\Rightarrow B \text{ in } P_1$ con $P \xrightarrow{C} P_1$ y $s' \in \text{tr}(A \not\Rightarrow B \text{ in } P_1)$. Por inducción, se obtiene que $[s'] \in \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P_1))$. Mediante la definición 3.4.1 existen dos casos:

- $A \notin [s']$: En este caso la traza $[s'] \in \text{prod}(P_1)$, por lo tanto $\{C\} \cup [s'] \in \text{prod}(P)$.

Desde que $A \notin \{C\} \cup [s']$, se cumple que

$$p = \{C\} \cup [s'] \in \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$$

- $B \notin [s']$: Este caso, es idéntico al caso anterior.

Ahora se demostrará que

$$\llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P)) \subseteq \text{prod}(A \not\Rightarrow B \text{ in } P)$$

Considerando $p \in \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P))$, existen dos casos:

- $A \notin p$: En este caso, $p \in \text{prod}(P)$. Entonces, existe una traza $s \in \text{tr}(P)$, tal que $p = [s]$. Es demostrado por inducción sobre la longitud de s .
 - $s = \epsilon$: En este caso, se tiene la transición $P \xrightarrow{\epsilon} \text{nil}$. Entonces, utilizando la regla [excl4], $A \not\Rightarrow B \text{ in } P \xrightarrow{\epsilon} \text{nil}$. De esta manera, $p = \emptyset \in \text{prod}(A \not\Rightarrow B \text{ in } P)$
 - $s = C \cdot s'$: Ahora, existen dos posibilidades: $C = B$ o $C \neq B$. Para el primer caso, se tiene la transición $P \xrightarrow{B} P_1$ con $s' \in \text{tr}(P_1)$. Utilizando la regla [excl3], se obtiene la transición $A \not\Rightarrow B \text{ in } P \xrightarrow{B} P_1 \setminus A$. Por un lado, se obtiene que $[s'] \in \text{prod}(P_1)$ y por el otro lado que $A \notin [s'] \subseteq p$, entonces $[s'] \in \llbracket \cdot \setminus A \rrbracket(P_1)$. Haciendo uso del lema 3.4.9 se obtiene que $[s'] \in \text{prod}(P_1 \setminus A)$, y entonces

$$p = \{C\} \cup [s'] = \{B\} \cup [s'] \in \text{prod}(A \not\Rightarrow B \text{ in } P)$$

Si $C \neq B$, se tiene la transición $P \xrightarrow{C} P_1$ con $s' \in \text{tr}(P_1)$. Utilizando la regla [excl1], se obtiene la transición $A \not\Rightarrow B \text{ in } P \xrightarrow{C} A \not\Rightarrow B \text{ in } P_1$. Desde que $A \notin [s']$, se obtiene que $[s'] \in \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket(\text{prod}(P_1))$. Por inducción, se obtiene que $[s'] \in \text{prod}(A \not\Rightarrow B \text{ in } P_1)$, entonces

$$p = \{C\} \cup [s'] \in \text{prod}(A \not\Rightarrow B \text{ in } P)$$

- $B \notin p$: Este caso es similar al anterior. La única diferencia, está en el caso inductivo cuando se observan las posibilidades del cómputo de la característica C . En este caso, las posibilidades son: $C = A$ y $C \neq A$. La segunda posibilidad es similar a la segunda opción del caso anterior. La diferencia con respecto a la primera, es que es necesario aplicar la regla [excl2] en vez de la regla [excl3].

□

Teorema 3.4.12 Dado el término $P \in \mathbf{SPLA}$, entonces $\text{prod}(P) = \llbracket P \rrbracket$.

Demostración: Será demostrado por inducción estructural sobre el término P , tomando en cuenta los lemas anteriores. Los casos base son $P = \text{nil}$ y $P = \checkmark$. En este caso, no es difícil comprobar que $\text{prod}(\text{nil}) = \llbracket \text{nil} \rrbracket = \emptyset$ y $\text{prod}(\checkmark) = \llbracket \checkmark \rrbracket = \{\emptyset\}$. La demostración por inducción se corresponde a:

- $P = A; P'$: Este caso corresponde al lema 3.4.4.
- $P = \bar{A}; P'$: Este caso corresponde al lema 3.4.5.
- $P = P_1 \vee P_2$: Este caso corresponde al lema 3.4.6.
- $P = P_1 \wedge P_2$: Este caso corresponde al lema 3.4.7.
- $P = A \Rightarrow B \text{ in } P$: Este caso corresponde al lema 3.4.10.
- $P = A \not\Rightarrow B \text{ in } P$: Este caso corresponde al lema 3.4.11.
- $P = P \setminus A$: Este caso corresponde al lema 3.4.9.
- $P = P \Rightarrow A$: Este caso corresponde al lema 3.4.8.

□

Una vez obtenidos los resultados buscados, es posible concluir que la semántica denotacional y la semántica operacional son equivalentes. El resultado inmediato de los resultados obtenidos en el teorema previo es que la relación de equivalencia \equiv es congruente.

Corolario 3.4.13 La relación de equivalencia \equiv es congruente. De esa forma, para cualquier operador n -ario op y $P_1, \dots, P_n, Q_1, \dots, Q_n \in \mathbf{SPLA}$, tal que $P_1 \equiv Q_1, \dots, P_n \equiv Q_n$, tenemos

$$\text{op}(P_1, \dots, P_n) \equiv \text{op}(Q_1, \dots, Q_n)$$

Demostración: Para realizar la demostración, solo es necesario considerar lo siguiente:

$$\text{prod}(\text{op}(P_1, \dots, P_n)) = \llbracket \text{op}(P_1, \dots, P_n) \rrbracket = \llbracket \text{op} \rrbracket(\llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket)$$

Desde que $P_i \equiv Q_i$, $\llbracket P_i \rrbracket = \text{prod}(P_i) = \text{prod}(Q_i) = \llbracket Q_i \rrbracket$. Entonces se obtiene

$$\llbracket \text{op} \rrbracket(\llbracket P_1 \rrbracket, \dots, \llbracket P_n \rrbracket) = \llbracket \text{op} \rrbracket(\llbracket Q_1 \rrbracket, \dots, \llbracket Q_n \rrbracket) = \llbracket \text{op}(Q_1, \dots, Q_n) \rrbracket = \text{prod}(\text{op}(Q_1, \dots, Q_n))$$

□

3.5. Reducción de los términos sintácticos

En esta sección se muestra que el modelo desarrollado es totalmente equivalente. Dado cualquier conjunto s de productos, existe un término SPLA cuya semántica es exacta al conjunto s . Este término puede ser construido utilizando subconjuntos de la gramática, tales como `nil`, `✓`, el operador de prefijo y el de selección única.

Definición 3.5.1 Dado un término $P \in \text{SPLA}$, se dice que P es un *término básico* si puede ser generado a partir de la siguiente gramática.

$$P ::= \checkmark \mid \text{nil} \mid A; P \mid P \vee Q$$

Se denota a este conjunto de términos básicos por SPLA_b .

□

Teorema 3.5.2 Dado un conjunto finito de características \mathcal{F} y $A \in \mathcal{P}(\mathcal{P}(\mathcal{F}))$, existe $P \in \text{SPLA}_b$, tal que $\text{prod}(P) = A$.

Demostración: Desde que \mathcal{F} es finito, entonces A es finito, este resultado será demostrado el resultado por inducción sobre $|A|$.

- $|A| = 0$: En este caso $P = \text{nil}$.
- $|A| > 0$: Considérese un producto $p \in A$ y el conjunto $A' = A \setminus \{p\}$. Por inducción, existe un producto $P' \in \text{SPLA}_b$, tal que $\text{prod}(P') = A'$. Desde que \mathcal{F} es finito, p es un conjunto finito de características $p = \{A_1, \dots, A_n\}$. Entonces, el término

$$P = (A_1; \dots; A_n; \checkmark) \vee P'$$

satisface la tesis del resultado.

□

Este resultado indica una congruencia práctica importante, de esta manera se puede suponer que cualquier operador puede ser agregado a la sintaxis de **SPLA**. Además, esto muestra que el operador puede ser *derivado* de los operadores de **SPLA_b**, así como los operadores en **SPLA** pueden ser reescritos en función de los operadores de **SPLA_b**.

El resultado establece una consecuencia teórica consistente. Una vez que la semántica denotacional es totalmente equivalente, también es isomorfa al modelo inicial con respecto a la relación de equivalencia. Eso significa que el conjunto de productos con los operadores de la semántica denotacional es isomorfa al álgebra de los términos, tal que SPLA/\equiv .

3.6. Semántica Axiomática

[REQ1] $A \Rightarrow B \text{ in } (C; P) =_E C; (A \Rightarrow B \text{ in } P)$	[EXCL1] $A \not\Rightarrow B \text{ in } (C; P) =_E C; (A \not\Rightarrow B \text{ in } P)$
[REQ2] $A \Rightarrow B \text{ in } (A; P) =_E A; (P \Rightarrow B)$	[EXCL2] $A \not\Rightarrow B \text{ in } (A; P) =_E A; (P \setminus B)$
[REQ3] $A \Rightarrow B \text{ in } (B; P) =_E B; P$	[EXCL3] $A \not\Rightarrow B \text{ in } (B; P) =_E B; (P \setminus A)$
[REQ4] $A \Rightarrow B \text{ in } P \vee Q =_E (A \Rightarrow B \text{ in } P) \vee (A \Rightarrow B \text{ in } Q)$	[EXCL4] $A \not\Rightarrow B \text{ in } P \vee Q =_E (A \not\Rightarrow B \text{ in } P) \vee (A \not\Rightarrow B \text{ in } Q)$
[REQ5] $A \Rightarrow B \text{ in } \checkmark =_E \checkmark$	[EXCL5] $A \not\Rightarrow B \text{ in } \checkmark =_E \checkmark$
[REQ6] $A \Rightarrow B \text{ in nil} =_E \text{nil}$	[EXCL6] $A \not\Rightarrow B \text{ in nil} =_E \text{nil}$
[MAND1] $(A; P) \Rightarrow A =_E A; P$	[FORB1] $(A; P) \setminus A =_E \text{nil}$
[MAND2] $(B; P) \Rightarrow A =_E B; (P \Rightarrow A)$	[FORB2] $(B; P) \setminus A =_E B; (P \setminus A)$
[MAND3] $\checkmark \Rightarrow A =_E A; \checkmark$	[FORB3] $\checkmark \setminus A =_E \checkmark$
[MAND4] $\text{nil} \Rightarrow A =_E \text{nil}$	[FORB4] $\text{nil} \setminus A =_E \text{nil}$
[MAND5] $(P \vee Q) \Rightarrow A =_E (P \Rightarrow A) \vee (Q \Rightarrow A)$	[FORB5] $(P \vee Q) \setminus A =_E (P \setminus A) \vee (Q \setminus A)$

Figura 3.8: Axiomas para eliminar los operadores de *requerimiento*, *obligación*, *exclusión* y *prohibición*.

En esta sección se presenta la semántica axiomática para **SPLA**, mostrando axiomas congruentes y completos para el lenguaje. Generalmente, la *congruencia* significa que las

[PRE1] $A; B; P =_E B; A; P.$	[CON1] $(A; P) \wedge Q =_E A; (P \wedge Q)$
[PRE2] $\bar{A}; P =_E (A; P) \vee \checkmark.$	[CON2] $P \wedge Q =_E Q \wedge P.$
[PRE3] $(A; P) \vee (A; Q) =_E A; (P \vee Q)$	[CON3] $P \wedge (Q \vee R) =_E (P \wedge Q) \vee (P \wedge R).$
[PRE4] $A; \text{nil} =_E \text{nil}$	[CON4] $P \wedge \text{nil} =_E \text{nil}$
[PRE5] $A; A; P =_E A; P$	[CON5] $P \wedge \checkmark =_E P$
[CHO1] $P \vee Q =_E Q \vee P.$	
[CHO2] $(P \vee Q) \vee R =_E P \vee (Q \vee R).$	
[CHO3] $P \vee \text{nil} =_E P.$	
[CHO4] $P \vee P = P.$	

Figura 3.9: Axiomas para eliminar los operadores *básicos*, *características opcionales* y el operador de *conjunción*.

igualdades deducidas de los axiomas son correctas, como por ejemplo, $P =_E Q$ implica que $P \equiv Q$. La *completitud* hace referencia a que todas las identidades pueden ser deducidas del sistema de axiomas, lo que significa que $P \equiv Q$ implica que $P =_E Q$.

Definición 3.6.1 Dados los términos $P, Q \in \text{SPLA}$, se dice que puede ser *deducida la equivalencia* de P y Q , si $P =_E Q$ puede ser deducido del conjunto de ecuaciones mostrado en la figura 3.8 y en la figura 3.9. \square

Para demostrar la congruencia es suficiente con mostrar que los operadores son congruentes (ver corolario 3.4.13) y que cada axioma es correcto.

Proposición 3.6.2 Dadas dos características $A, B \in \mathcal{F}$ y dos términos SPLA P y Q , las ecuaciones mostradas en la figura 3.8 y en la figura 3.9 son correctas.

Demostración: En todos los axiomas, a excepción del axioma [PRE1], el axioma [PRE4], el axioma [PRE5], el axioma [REQ3], el axioma [CON1] y el axioma [CON4], las transiciones de los términos en ambos lados de la ecuación son iguales. Entonces, en estos casos las trazas son las mismas y tienen los mismos productos.

En los otros casos, la demostración es realizada utilizando la semántica denotacional.

- **[PRE1]:** $A; B; P =_E B; A; P$

$$\begin{aligned} \llbracket A; B; P \rrbracket &= \{A\} \cup \llbracket B; P \rrbracket = \{A\} \cup \{B\} \cup \llbracket P \rrbracket = \\ &\{B\} \cup \{A\} \cup \llbracket P \rrbracket = \{B\} \cup \llbracket A; P \rrbracket = \llbracket B; A; P \rrbracket \end{aligned}$$

- **[PRE4]:** $A; \text{nil} =_E \text{nil}$

$$\begin{aligned} p \in \llbracket A; \text{nil} \rrbracket &\iff \exists p' \in \llbracket \text{nil} \rrbracket \wedge p = p' \cup \{A\} \\ &\iff \text{Falso} \iff p \in \llbracket \text{nil} \rrbracket \end{aligned}$$

- **[PRE5]:** $A; A; P =_E A; P$

$$\begin{aligned} \llbracket A; A; P \rrbracket &= \{A\} \cup \llbracket A; P \rrbracket = \{A\} \cup \{A\} \cup \llbracket P \rrbracket = \\ &\{A\} \cup \llbracket P \rrbracket = \llbracket A; P \rrbracket \end{aligned}$$

- **[REQ3]:** $A \Rightarrow B \text{ in } (B; P) =_E B; P$. En este caso, es suficiente con considerar que $B \in \llbracket B; P \rrbracket$. Por lo tanto, utilizando la definición 3.4.1, se obtiene que

$$p \in \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket (\llbracket B; P \rrbracket) \iff p \in \llbracket B; P \rrbracket$$

- **[CON1]:** $(A; P) \wedge Q =_E A; (P \wedge Q)$

$$\begin{aligned} p \in \llbracket (A; P) \wedge Q \rrbracket &\iff \\ \exists p_1 \in \llbracket A; P \rrbracket, p_2 \in \llbracket Q \rrbracket : p &= p_1 \cup p_2 \iff \\ \exists p'_1 \in \llbracket P \rrbracket, p_2 \in \llbracket Q \rrbracket : p &= p'_1 \cup \{A\} \cup p_2 \iff \\ \exists p' \in \llbracket P \wedge Q \rrbracket : p &= p' \cup \{A\} \iff \\ p \in \llbracket A; (P \wedge Q) \rrbracket & \end{aligned}$$

- **[CON4]:** $P \wedge \text{nil} =_E \text{nil}$

$$\begin{aligned} p \in \llbracket P \wedge \text{nil} \rrbracket &\iff \\ \exists p_1 \in \llbracket P \rrbracket, p_2 \in \llbracket \text{nil} \rrbracket : p &= p_1 \cup p_2 \iff \\ \exists p_1 \in \llbracket P \rrbracket, p_2 \in \emptyset : p &= p_1 \cup p_2 \iff \\ \text{Falso} &\iff p \in \llbracket \text{nil} \rrbracket \end{aligned}$$

□

Para demostrar la completitud es necesario el concepto de las *formas normales*. De esta manera, es posible mostrar que los operadores sintácticos son *derivados* de los operadores *básicos*. Los operadores básicos son nil , \checkmark , el operador del prefijo $(A; P)$ y el operador de *selección única* $(P \vee Q)$. A continuación se muestra un ejemplo en el que se muestra cómo algunos operadores pueden ser eliminados. Teniendo en cuenta la siguiente SPL, $P = A; \checkmark \wedge B; \checkmark$, es fácil generar las trazas satisfactorias, siendo éstas $\{AB, BA\}$. Y $\text{prod}(P) = \{[AB]\}$. Esta SPL tiene los mismos productos que $A; B; \checkmark$. Así, aplicando los axiomas correspondientes, es posible realizar la siguiente deducción:

$$\begin{aligned}
& A; \checkmark \wedge B; \checkmark =_E \quad [\text{CON1}] \\
& A; (\checkmark \wedge B; \checkmark) =_E \quad [\text{CON2}] \\
& A; (B; \checkmark \wedge \checkmark) =_E \quad [\text{CON1}] \\
& A; B; (\checkmark \wedge \checkmark) =_E \quad [\text{CON5}] \\
& A; B; \checkmark
\end{aligned}$$

El conjunto de axiomas mostrados en la figura 3.8 y en la figura 3.9 permite que los operadores no básicos (ver definición 3.5.1) puedan ser eliminados de cualquier $P \in \text{SPLA}$. El objetivo es poder demostrar que existe un término $Q \in \text{SPLA}_b$ tal que $P \equiv Q$.

Suponemos que se tiene un término $P \in \text{SPLA}$ que contiene un operador no básico. Entonces es posible encontrar otro término $Q \in \text{SPLA}$ equivalente a P , donde, o el *operador no básico* ha desaparecido o está a una mayor profundidad dentro del árbol sintáctico de Q . Así, al iterar sobre este proceso es posible permitir que desaparezcan todos los operadores no básicos, obteniendo el teorema buscado.

Teorema 3.6.3 Dado el término $P \in \text{SPLA}$, existe un término $Q \in \text{SPLA}_b$ tal que $P =_E Q$.

Demostración: La demostración es trivial por inducción estructural sobre P .

□

Puesto que es posible eliminar aquellos operadores no básicos de un término **SPLA**, el estudio se enfocará en demostrar la completitud restringida a los términos básicos. Por ello, se definen las *formas normales* para poder demostrar que cualquier término básico puede ser transformado a la forma normal utilizando los axiomas **PRE1**, **PRE2**, **PRE3**, **PRE4**, **PRE5**, **CHO1**, **CHO2**, **CHO3** y **CHO4**, descritos en la figura 3.9.

Para definir formalmente la forma normal, previamente se deben proporcionar algunas definiciones auxiliares. Primero, se asume que existe una relación de orden $\leq \subseteq \mathcal{F} \times \mathcal{F}$, la cual debe ser isomorfa a los números naturales en los casos en los que \mathcal{F} sea infinito. Segundo, es necesario definir el *vocabulario* de un término básico **SPLA**, el cual se corresponde con el conjunto de características que aparecen en la expresión.

Definición 3.6.4 Dados dos términos básicos $P, Q \in \mathbf{SPLA}_b$, se define el *vocabulario* de **SPLA** como la función $\text{voc} : \mathbf{SPLA}_b \rightarrow \mathcal{P}(\mathcal{F})$ definida inductivamente como:

- $\text{voc}(\text{nil}) = \text{voc}(\checkmark) = \emptyset$
- $\text{voc}(A; P) = \{A\} \cup \text{voc}(P)$
- $\text{voc}(P \vee Q) = \text{voc}(P) \cup \text{voc}(Q)$

□

Con el fin de proporcionar de forma clara la definición de las formas normales, se define previamente un caso más simple, las formas pre-normales.

Definición 3.6.5 Un término básico **SPLA** $P \in \mathbf{SPLA}_b$ está en *forma pre-normal*, denotado por $P \in \mathbf{SPLA}_{\text{pre}}$, si y solo si tiene una de las siguientes formas.

1. nil, \checkmark

ó

2. Existe $n > 0$, $\{A_1, \dots, A_n\} \subseteq \mathcal{F}$ y existe $P_1, \dots, P_n \in \mathbf{SPLA}_{\text{pre}}$ con $P_i \neq \text{nil}$ para $1 \leq i \leq n$ y $\{A_1, \dots, A_n\} \cap \text{voc}(P_j) = \emptyset$ para $1 \leq j \leq n$ y también

$$P = (A_1; P_1) \vee \dots \vee (A_n; P_n) \quad \text{ó} \quad P = (A_1; P_1) \vee \dots \vee (A_n; P_n) \vee \checkmark$$

en cuyo caso se dice que las características $\{A_1, \dots, A_n\}$ están en el primer nivel de P . \square

Seguidamente se presenta un lema auxiliar que será utilizado en la proposición 3.6.12. Este lema establece que si una característica aparece en el vocabulario de una forma pre-normal, entonces aparece en al menos un producto de la forma pre-normal. Es importante destacar que este resultado no es cierto³ en términos ordinarios debido a las restricciones que pueden aparecer en los términos. Por ejemplo, al utilizar la restricción de ocultamiento ya que la característica puede pertenecer al vocabulario, pero, debido a estar oculta no estaría presente en los productos válidos.

Lema 3.6.6 Dado un término $P \in \text{SPLA}_{\text{pre}}$, entonces

$$\text{voc}(P) = \{A \mid A \in p, p \in \text{prod}(P)\}$$

Demostración: La demostración es realizada por inducción sobre la profundidad de P . Nótese, que cada término P_i que aparece en la parte 2 de la definición 3.6.5 no puede ser `nil`. \square

El siguiente lema establece que si una característica A es descrita como una forma pre-normal P , ésta puede ser transformada en otra forma pre-normal equivalente Q tal que A esté en el primer nivel del árbol sintáctico de Q .

Lema 3.6.7 Dado un término $P \in \text{SPLA}_{\text{pre}}$ y una característica $A \in \text{voc}(P)$, entonces existe un término $Q \in \text{SPLA}_{\text{pre}}$, tal que $P \equiv Q$ y $A \in \{A_1, \dots, A_n\}$ de acuerdo a la condición 2 de la definición 3.6.5 aplicada a Q .

Demostración: La demostración es realizada por inducción estructural de P al aplicar el axioma **[PRE1]** y el axioma **[PRE3]**. \square

³El vocabulario de un término ordinario no ha sido formalmente definido. La definición 3.6.4 puede ser extendida al conjunto de características presentes en la sintaxis de un término.

La siguiente proposición establece el primer resultado necesario para demostrar la completitud, donde cualquier término puede ser transformado a una forma pre-normal equivalente. El resultado está restringido a términos básicos, sin embargo, por el teorema 3.6.3, puede ser extendido a cualquier otro término ordinario.

Proposición 3.6.8 Dado un término $P \in \text{SPLA}_b$, existe una forma pre-normal $Q \in \text{SPLA}_{\text{pre}}$, tal que $P =_E Q$.

Demostración: La demostración es realizada por inducción estructural de P y al aplicar el lema 3.6.7.

□

El inconveniente de utilizar las formas pre-normales es que existen expresiones sintácticamente diferentes que son equivalentes, tal y como muestra el siguiente ejemplo.

Ejemplo 3.6.9 Dadas las siguientes expresiones SPLA_{pre} :

$$\begin{aligned} P &= (A; C; \checkmark) \vee B; \checkmark \\ Q &= (C; A; \checkmark) \vee B; \checkmark \end{aligned}$$

Ambas expresiones están en forma pre-normal y son equivalentes.

La manera de obtener una única forma normal, para cualquier expresión SPLA_b , consiste en utilizar el orden antes mencionado entre las características. Asumiendo $A < B < C$, en este caso, se puede decir que P está en forma normal, mientras que Q no lo está. □

Las formas normales son un caso particular de las formas pre-normales, las cuales hacen uso del orden requerido entre las características.

Definición 3.6.10 Dado un término $P \in \text{SPLA}_{\text{pre}}$, se dice que P es una *forma normal*, denotada por $P \in \text{SPLA}_{\text{nf}}$, si y solo si $P = \text{nil}$, $P = \checkmark$ o si los conjuntos $\{A_1, \dots, A_n\}$ y $\{P_1, \dots, P_n\}$, según el apartado 2 de la definición 3.6.5 satisfacen:

- $A_i < A_j$ para $1 \leq i < j \leq n$.

- $A_i < B$ para cada $B \in \text{voc}(P_j)$ y para $1 \leq i \leq j \leq n$.

□

Las formas normales correspondientes a los ejemplos de la figura 3.2 se muestran en la figura 3.10 y en la figura 3.11, asumiendo que $A < B < C$. Los ejemplos **a**, **b** y **c** no están incluidos, ya que son formas normales.

d	e
$A; (B; \checkmark \wedge C; \checkmark) =_E \quad [\text{CON1}]$	$A; (\bar{B}; \checkmark \wedge C; \checkmark) =_E \quad [\text{CON2}], \quad [\text{CON1}]$
$A; B; (\checkmark \wedge C; \checkmark) =_E \quad [\text{CON2}], \quad [\text{CON5}]$	$A; C; (\checkmark \wedge \bar{B}; \checkmark) =_E \quad [\text{CON2}], \quad [\text{CON5}]$
$A; B; C; \checkmark$	$A; C; \bar{B}; \checkmark =_E \quad [\text{PRE2}]$
	$A; C; (B; \checkmark \vee \checkmark) =_E \quad [\text{PRE3}]$
	$A; (C; B; \checkmark \vee C; \checkmark) =_E \quad [\text{PRE1}]$
	$A; (B; C; \checkmark \vee C; \checkmark)$

f

$$\begin{aligned}
 & A; (\bar{B}; \checkmark \wedge \bar{C}; \checkmark) =_E \quad [\text{PRE2}] \\
 & A; ((B; \checkmark \vee \checkmark) \wedge (C; \checkmark \vee \checkmark)) =_E \quad [\text{CON3}] \\
 & A; ((B; \checkmark \wedge C; \checkmark) \vee \\
 & (B; \checkmark \wedge \checkmark) \vee (\checkmark \wedge C; \checkmark) \vee (\checkmark \wedge \checkmark)) =_E \quad [\text{CON1}] \\
 & \quad \quad \quad [\text{CON5}] \\
 & A; ((B; C; \checkmark) \vee (B; \checkmark) \vee (C; \checkmark) \vee \checkmark)
 \end{aligned} \tag{3.1}$$

$$\begin{aligned}
 & B \not\leq C \text{ in } A; (\bar{B}; \checkmark \wedge \bar{C}; \checkmark) =_E \quad (3.1) \\
 & A; B \not\leq C \text{ in } \left(\begin{array}{l} (B; C; \checkmark) \vee \\ (B; \checkmark) \vee (C; \checkmark) \vee \checkmark \end{array} \right) =_E \quad [\text{EXCL4}] \\
 & A; \left(\begin{array}{l} (B \not\leq C \text{ in } B; C; \checkmark) \vee \\ (B \not\leq C \text{ in } B; \checkmark) \vee \\ (B \not\leq C \text{ in } C; \checkmark) \vee \\ (B \not\leq C \text{ in } \checkmark) \end{array} \right) =_E \quad \begin{array}{l} [\text{EXCL2}] \\ [\text{EXCL3}] \\ [\text{EXCL5}] \end{array} \\
 & A; \left(\begin{array}{l} (B; (C; \checkmark \setminus C)) \vee \\ (B; (\checkmark \setminus C)) \vee (C; (\checkmark \setminus B)) \vee \checkmark \end{array} \right) =_E \quad \begin{array}{l} [\text{FORB1}] \\ [\text{FORB3}] \end{array} \\
 & A; ((B; \text{nil}) \vee (B; \checkmark) \vee (C; \checkmark) \vee \checkmark) =_E \quad [\text{PRE4}] \\
 & A; (\text{nil} \vee (B; \checkmark) \vee (C; \checkmark) \vee \checkmark) =_E \quad [\text{CHO3}] \\
 & A; ((B; \checkmark) \vee (C; \checkmark) \vee \checkmark)
 \end{aligned}$$

Figura 3.10: Transformación a forma normal para los ejemplos **d**, **e** y **f**.

$$\begin{aligned}
& \mathbf{g} \\
& B \Rightarrow C \text{ in } A; (\bar{B}; \checkmark \wedge \bar{C}; \checkmark) =_E \quad (3.1) \\
& A; B \Rightarrow C \text{ in } \left(\begin{array}{l} (B; C; \checkmark) \vee \\ (B; \checkmark) \vee (C; \checkmark) \vee \checkmark \end{array} \right) =_E \quad [\text{REQ4}] \\
& A; \left(\begin{array}{l} (B \Rightarrow C \text{ in } B; C; \checkmark) \vee \\ (B \Rightarrow C \text{ in } B; \checkmark) \vee \\ (B \Rightarrow C \text{ in } C; \checkmark) \vee \\ (B \Rightarrow C \text{ in } \checkmark) \end{array} \right) =_E \quad \begin{array}{l} [\text{REQ2}] \\ [\text{REQ3}] \\ [\text{REQ5}] \end{array} \\
& A; \left(\begin{array}{l} (B; (C; \checkmark \Rightarrow C)) \vee \\ (B; (\checkmark \Rightarrow C)) \vee (C; \checkmark) \vee \checkmark \end{array} \right) =_E \quad \begin{array}{l} [\text{MAND1}] \\ [\text{MAND3}] \end{array} \\
& A; \left(\begin{array}{l} (B; C; \checkmark) \vee \\ (B; C; \checkmark) \vee (C; \checkmark) \vee \checkmark \end{array} \right) =_E \quad [\text{CHO4}] \\
& A; (B; C; \checkmark \vee C; \checkmark \vee \checkmark)
\end{aligned}$$

Figura 3.11: Transformación a forma normal para el ejemplo **g**.

Utilizando las definiciones anteriores, es posible transformar cualquier expresión en forma normal.

Proposición 3.6.11 Dado el término $P \in \text{SPLA}_{\text{pre}}$, entonces existe una forma normal $Q \in \text{SPLA}_{\text{nf}}$ tal que $P =_E Q$.

Demostración: La demostración es realizada por inducción sobre la profundidad de la forma pre-normal P . El caso base, es trivial porque $P = \checkmark$ o $P = \text{nil}$, para estos casos, P está en forma normal.

El caso inductivo necesita ser detallado. La diferencia entre la forma normal y la forma pre-normal es el orden impuesto a las características. Básicamente existen dos casos. El primer caso, es cuando P tiene la siguiente forma:

$$P = \dots \vee B; P \vee \dots \vee A; Q \vee \dots \quad \text{con } A < B$$

Este término puede ser transformado en forma normal al aplicar la conmutatividad del operador de *selección única* (ecuación **[CHO1]**).

El segundo caso, es cuando las características que no están propiamente ordenadas aparecen en el mismo subtérmino:

$$P = \dots \vee B; (\dots A; P \dots) \vee \dots \quad \text{con } A < B$$

Este caso, puede ser transformado al aplicar las ecuaciones **[PRE3]** y **[PRE1]** como se indica a continuación:

$$\begin{array}{ll} B; ((A; P) \vee Q) =_E & \text{[PRE3]} \\ (B; A; P) \vee (B; Q) =_E & \text{[PRE1]} \\ (A; B; P) \vee (B; Q) & \end{array}$$

□

La siguiente proposición muestra que dos formas normales que son semánticamente equivalentes, también lo son a nivel léxico.

Proposición 3.6.12 Dados los términos $P, Q \in \text{SPLA}_{\text{nf}}$, si éstos son semánticamente equivalentes, tal que $P \equiv Q$, entonces también son sintácticamente idénticas, de forma que $P = Q$.

Demostración: La demostración será realizada por contradicción. Suponemos que se tienen los términos $P, Q \in \text{SPLA}_{\text{nf}}$ y que estos son sintácticamente distintos. Será demostrado que estos no son equivalentes $P \not\equiv Q$. La demostración es realizada por inducción sobre la profundidad de P y Q . El caso base, es cuando ambos son \checkmark o nil y el resultado es trivial. Para el caso inductivo, se tienen las siguientes posibilidades:

- (a) $P = (A_1; P_1) \vee (A_m; P_n)$,
- (b) $P = (A_1; P_1) \vee (A_m; P_n) \vee \checkmark$,
- (c) $Q = (B_1; Q_1) \vee (B_m; Q_m)$, o
- (d) $Q = (B_1; Q_1) \vee (B_m; Q_m) \vee \checkmark$

Para el caso (a)+(d) $\emptyset \in \text{prod}(P)$ donde $\notin \text{prod}(Q)$, entonces $P \not\equiv Q$. El caso (b)+(c) es simétrico. Considérese el caso (a)+(c), el caso (b)+(d) es resuelto de la misma manera. Si P y Q son sintácticamente diferentes, entonces existen dos posibilidades:

- $\{A_1, \dots, A_n\} \neq \{B_1, \dots, B_m\}$: Considérese la primera diferencia entre ambos conjuntos. Se debe asumir que la primera diferencia consiste en un elemento del primer conjunto que no se encuentra en el segundo, lo que es, que existe un elemento $k \in \{1, \dots, n\}$, tal que $A_k \notin \{B_1, \dots, B_m\}$ y $A_i = B_i$ para $i < k$. Entonces, existen las siguientes posibilidades:
 - $k = m + 1$: Cualquier ocurrencia de A_k en Q debe estar en cualquiera de los sub-árboles Q_j , $j < k$. Empleando el lema 3.6.6, se obtiene que $B_j \in p$ para cualquier producto $p \in \text{prod}(Q)$, tal que $A_k \in P$. Sin embargo $B_j < A_k$, entonces $B_j \notin \text{voc}(P_k)$. Por lo tanto, existen productos $p' \in \text{prod}(P)$, tal que $B_j \notin p'$. De esta manera $P \not\equiv Q$.
 - $A_k < B_k$: Este caso es similar al anterior, porque $A_k \neq B_l$ para $l \geq k$ y $A_k \notin \text{voc}(Q_l)$ para $l \geq k$.
 - $A_k > B_k$: Este caso es simétrico al anterior.
- $n = m$, $A_i = B_i$ para $1 \leq i \leq n$ y $\{P_1, \dots, P_n\} \neq \{Q_1, \dots, Q_n\}$: Considérese el primer k tal que $P_k \neq Q_k$. Por inducción estructural se tiene que $P_k \not\equiv Q_k$, asumiendo que $p \in \text{prod}(P_k)$ pero $p \notin \text{prod}(Q_k)$. Es claro que $p \cup \{A_k\} \in \text{prod}(P)$, ahora se demostrará que $p \cup \{A_k\} \notin \text{prod}(Q)$. Por otro lado, desde que $A_k \notin \text{voc}(Q_i)$ para $i \geq k$, al utilizar el lema 3.6.6, $p \cup \{A_k\} \notin \text{prod}(A_i; Q_i)$ para $i \geq k$. Por otro lado, desde que $A_i \notin \text{voc}(P_k)$ para $i < k$, nuevamente utilizando el lema 3.6.6, $A_i \notin p$ y así $A_i \notin p \cup \{A_k\}$ si $i < k$. Desde que $A_i \in q$ para cualquier $q \in \text{prod}(A_i; Q_i)$, $p \cup \{A_k\} \notin \text{prod}(A_i; Q_i)$ para $i < k$. Así que, para cualquier $1 \leq i \leq n$, $p \cup \{A_k\} \notin \text{prod}(A_i; Q_i)$ y entonces $p \cup \{A_k\} \notin \text{prod}(Q)$. Por lo tanto $P \not\equiv Q$.

□

Finalmente, es posible llevar a cabo el objetivo principal de esta sección, el cual consiste en demostrar que el sistema deductivo es consistente y completo.

Teorema 3.6.13 Dados los términos $P, Q \in \text{SPLA}$, entonces $P \equiv Q$, si y solo si $P =_E Q$.

Demostración: Deben ser demostradas dos implicaciones:

- **Si $P =_E Q$, entonces $P \equiv Q$:** Esto es consecuencia de la consistencia de cada regla (proposición 3.6.2).
- **Si $P \equiv Q$, entonces $P =_E Q$:** Utilizando la proposición 3.6.8, existen $P_{pre}, Q_{pre} \in \text{SPLA}_{pre}$, tal que $P_{pre} =_E P$ y $Q =_E Q_{pre}$. Ahora, por la proposición 3.6.11, existen $P_{nf}, Q_{nf} \in \text{SPLA}_{nf}$, tal que $P_{nf} =_E P_{pre} =_E P$ y $Q_{nf} =_E Q_{pre} =_E Q$. Utilizando la proposición 3.6.2, obtenemos que

$$P_{nf} \equiv P_{pre} \equiv P \equiv Q \equiv Q_{pre} \equiv Q_{nf}$$

Finalmente, mediante el uso de la proposición 3.6.12, se obtiene que P_{nf} y Q_{nf} son idénticos así que $P =_E Q$.

□

3.7. Consistencia en la traducción de FODA a SPLA

El proceso de traducción descrito en la sección 3.2 no impone un orden para el procesamiento de las restricciones. Esto significa que un diagrama FODA puede generar distintos términos SPLA. Además, se demostrará que todos los términos, aparentemente distintos, son equivalentes. En la traducción, primero se procesan las restricciones de *requerimiento* del diagrama FODA. Seguidamente, cuando todas estas restricciones han sido procesadas, se procesan las de *exclusión*. En esta sección se demostrará, inicialmente, la propiedad que indica el orden en el que deben seleccionarse las restricciones de *requerimiento*. Es importante destacar que para realizar la traducción de FODA a SPLA se considerará el cierre de las restricciones de requerimiento.

Definición 3.7.1 Dado un término $P \in \text{SPLA}$, se dice que P está *cerrado con respecto a la restricción de requerimiento*, si tiene la siguiente forma:

$$A_1 \Rightarrow B_1 \text{ in } A_2 \Rightarrow B_2 \text{ in } \cdots A_n \Rightarrow B_n \text{ in } Q$$

donde Q no tiene restricciones y el conjunto de restricciones está cerrado por transitividad. Esto es, si existen características $A, B, C \in \mathcal{F}$ y $1 \leq i, j \leq n$, tal que $A = A_i$, $B = B_i = A_j$ y $C = B_j$, entonces existe $1 \leq k \leq n$ tal que $A = A_k$ y $C = B_k$.

Demostración: Considerando las características $A, B, C, D \in \mathcal{F}$. Si $A \neq D$, no es difícil comprobar utilizando la definición del operador de *requerimiento* descrito en la definición 3.4.1, lo siguiente

$$\llbracket A \Rightarrow B \text{ in } C \Rightarrow D \text{ in } P \rrbracket = \llbracket C \Rightarrow D \text{ in } A \Rightarrow B \text{ in } P \rrbracket$$

Sin embargo, si $A = D$, porque el término es cerrado entorno a la restricción de *requerimiento*, al utilizar la definición del operador de *requerimiento* se obtiene:

$$\begin{aligned} \llbracket A \Rightarrow B \text{ in } C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } P \rrbracket &= \\ \llbracket A \Rightarrow B \text{ in } C \Rightarrow A \text{ in } C \Rightarrow B \text{ in } P \rrbracket &= \\ \llbracket C \Rightarrow A \text{ in } A \Rightarrow B \text{ in } C \Rightarrow B \text{ in } P \rrbracket &= \\ \llbracket C \Rightarrow A \text{ in } C \Rightarrow B \text{ in } A \Rightarrow B \text{ in } P \rrbracket &= \\ \llbracket C \Rightarrow B \text{ in } A \Rightarrow B \text{ in } C \Rightarrow A \text{ in } P \rrbracket &= \\ \llbracket C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } A \Rightarrow B \text{ in } P \rrbracket & \end{aligned}$$

Será demostrado en detalle que

$$\llbracket A \Rightarrow B \text{ in } C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } P \rrbracket \subseteq \llbracket C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } A \Rightarrow B \text{ in } P \rrbracket$$

Los demás casos son similares, donde cualquier producto $p \in \llbracket A \Rightarrow B \text{ in } C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } P \rrbracket$, es construido a partir de un producto $p' \in \llbracket P \rrbracket$. Es posible distinguir los siguientes casos:

- $C \in p'$: Entonces, $p = p' \cup \{A, B\}$ y $p \in \llbracket C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } A \Rightarrow B \text{ in } P \rrbracket$.
- $C \notin p', A \in p'$: Entonces, $p = p' \cup \{B\}$ y $p \in \llbracket C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } A \Rightarrow B \text{ in } P \rrbracket$.

- $C, A \notin p'$: Entonces, $p = p'$ y $p \in \llbracket C \Rightarrow B \text{ in } C \Rightarrow A \text{ in } A \Rightarrow B \text{ in } P \rrbracket$.

□

Proposición 3.7.2 Dado un término $P \in \text{SPLA}$, el cual se considera cerrado con respecto a la restricción de *requerimiento* y el término $Q \in \text{SPLA}$, tal que puedan ser reorganizadas sus restricciones de *requerimiento* en P , para estas condiciones $P \equiv Q$. □

También se muestra que el orden en el cual las restricciones de *exclusión* son seleccionadas es irrelevante. Esto es debido a que, para todos los casos, siempre dos restricciones de *exclusión* en FODA son intercambiables.

Proposición 3.7.3 Dado un término $P \in \text{SPLA}$ y las características $A, B, C, D \in \mathcal{F}$, entonces $A \not\Rightarrow B \text{ in } C \not\Rightarrow D \text{ in } P \equiv C \not\Rightarrow D \text{ in } A \not\Rightarrow B \text{ in } P$.

Demostración: Esta proposición es inmediata por la definición del operador de *exclusión* en la definición 3.4.1. □

□

3.8. Factibilidad de los términos SPLA

Esta sección presenta un mecanismo para comprobar la *factibilidad* de un término sintáctico, esto es, determinar si existe algún producto que satisfaga todas las restricciones del término.

Definición 3.8.1 Dado un término $P \in \text{SPLA}$, se dice que P es *factible* si y solo si $\text{prod}(P) \neq \emptyset$. □

La comprobación de la factibilidad de cualquier término $P \in \text{SPLA}$ puede llevarse a cabo procesando todos sus productos, haciendo uso de las reglas definidas en la semántica operacional o en la semántica denotacional. Una vez hecho esto, es posible comprobar si es vacío o no el conjunto de productos válidos. Sin embargo, procesar todos los productos

válidos del modelo de variabilidad puede no ser posible. Dada esta condición al procesar los modelos de variabilidad, es necesario presentar una alternativa que permita determinar si un termino es factible o no. Para esto, se presenta una solución planteando el modelo de variabilidad como un problema de factibilidad lógica, implementado con el uso de *SAT solvers*. De cualquier término $P \in \mathbf{SPLA}$ se construye una fórmula proposicional $\phi(P)$, tal que P es factible, si y solo si, existe una evaluación v tal que $v \models \phi(P)$.

Al construir esta fórmula se debe mantener el orden en el que las características son producidas. Cualquier característica \mathbf{A} está asociada con un conjunto de variables lógicas \mathbf{A}_k , $k \in \mathbb{N}$, del cual, el entero asociado a la característica se utiliza para mantener el orden en que se ha procesado. Así, las variables lógicas tendrán la forma \mathbf{A}_k donde $\mathbf{A} \in \mathcal{F}$ y $k \in \mathbb{N}$.

Antes de describir cómo procesar la fórmula asociada al término sintáctico, es necesario presentar algunas definiciones auxiliares. La función **maxin** es relevante, ya que será utilizada para, dada una fórmula proposicional, procesar el siguiente índice disponible para una característica.

Definición 3.8.2 Dada una fórmula proposicional φ , se denota *el conjunto de variables lógicas en φ* como $\mathbf{vars}(\varphi)$.

Dada la característica $\mathbf{A} \in \mathcal{F}$ y la fórmula proposicional φ , se define la función que retorna el *máximo índice de \mathbf{A} en la fórmula φ* como:

$$\mathbf{maxin}(\mathbf{A}, \varphi) = \begin{cases} k & \exists l \in \mathbb{N} : \mathbf{A}_l \in \mathbf{vars}(\varphi), \\ & k = \text{máx}\{l \mid \mathbf{A}_l \in \mathbf{vars}(\varphi)\} \\ -1 & \text{de otra manera} \end{cases}$$

Finalmente, si $l < 0$, \mathbf{A}_l será denotado con el símbolo \perp . □

El lema 3.8.6 es indispensable para demostrar la factibilidad de los términos, ya que es necesario completar las *fórmulas* procesadas para que incluyan el operador de selección. Es conveniente que la función **maxin** sea la misma en ambas partes del operador de *selección única*, lo cual se consigue mediante la definición 3.8.3.

Definición 3.8.3 Dadas las fórmulas proposicionales φ_1 y φ_2 , se define la *completitud de φ_1 hasta φ_2* , escrito $\text{comp}(\varphi_1)\varphi_2$, de la siguiente manera:

$$\text{comp}(\varphi_1)\varphi_2 = \varphi_1 \wedge \bigwedge_{\substack{\mathbf{A} \in \mathcal{F}, \\ l = \text{maxin}(\mathbf{A}, \varphi_2), \\ k = \text{maxin}(\mathbf{A}, \varphi_1), \\ 0 \leq k < l}} (\neg \mathbf{A}_l \rightarrow \neg \mathbf{A}_{l-1}) \wedge \cdots \wedge (\neg \mathbf{A}_{k+1} \rightarrow \neg \mathbf{A}_k)$$

□

La primera consecuencia de la definición anterior es que $\text{comp}(\varphi_1)\varphi_2$ es más fuerte que φ_1 . Si $v \models \text{comp}(\varphi_1)\varphi_2$ entonces $v \models \varphi_1$.

Es relevante destacar que en la definición anterior, las variables nuevas no pertenecen a φ_1 . Entonces, cualquier evaluación v tal que $v \models \varphi_1$ puede ser extendida a una nueva evaluación v' de manera que únicamente modifique el valor de las nuevas variables y $v' \models \text{comp}(\varphi_1)\varphi_2$. También se puede destacar que el número de variables no se incrementa debido a esta completitud, lo cual sucede debido a que las variables que son introducidas en $\text{comp}(\varphi_1)\varphi_2$ ya están en φ_2 . Estas propiedades están expresadas en el lema 3.8.4.

Lema 3.8.4 Dadas las fórmulas proposicionales φ_1 y φ_2 .

1. Dada una evaluación v tal que $v \models \text{comp}(\varphi_1)\varphi_2$, entonces, $v \models \varphi_1$.
2. Dada una evaluación v tal que $v \models \varphi_1$, entonces, existe una evaluación v' tal que $v' \models \text{comp}(\varphi_1)\varphi_2$ y $v'(\mathbf{A}_l) = v(\mathbf{A}_l)$, para cualquier característica \mathbf{A} y $0 \leq l \leq \text{maxin}(\mathbf{A}, \varphi_1)$.
3. Dada una característica \mathbf{A} , tal que exista $k \in \mathbb{N}$ satisfaciendo $\mathbf{A}_k \in \text{vars}(\text{comp}(\varphi_1)\varphi_2)$ y $\mathbf{A}_k \notin \text{vars}(\varphi_1)$, entonces, $\mathbf{A}_k \in \text{vars}(\varphi_2)$. De esta manera, $\text{maxin}(\mathbf{A}, \text{comp}(\varphi_1)\varphi_2) = \text{maxin}(\mathbf{A}, \varphi_2)$

Demostración: A continuación se procederá a demostrar cada caso:

1. Por la construcción de $\text{comp}(\varphi_1)\varphi_2$.

2. Es suficiente con considerar la evaluación v' definida como:

$$v'(\mathbf{A}_l) = \begin{cases} v(\mathbf{A}_k) & \text{si } 0 \leq \text{maxin}(\mathbf{A}_l, \varphi) < k \\ v(\mathbf{A}_l) & \text{en otro caso} \end{cases}$$

3. Por construcción de $\text{comp}(\varphi_1)\varphi_2$.

□

Definición 3.8.5 Dado un término $P \in \text{SPLA}$, se define su fórmula proposicional asociada $\phi(P)$, como:

$$\begin{aligned} \phi(\text{nil}) &= \perp \\ \phi(\checkmark) &= \top \\ \phi(\mathbf{A}; P) &= \mathbf{A}_{l+1} \wedge \phi(P) \\ &\quad \text{donde } l = \text{máx}(0, \text{maxin}(\mathbf{A}, \phi(P))) \\ \phi(\bar{\mathbf{A}}; P) &= \top \\ \phi(P \vee Q) &= \text{comp}(\phi(P))\phi(Q) \vee \text{comp}(\phi(Q))\phi(P) \\ \phi(P \wedge Q) &= \phi(P) \wedge \phi(Q) \\ \phi(\mathbf{A} \Rightarrow \mathbf{B} \text{ in } P) &= (\neg \mathbf{A}_{l+1} \rightarrow \neg \mathbf{A}_l) \wedge (\neg \mathbf{B}_{m+1} \rightarrow \neg \mathbf{B}_m) \wedge \\ &\quad (\mathbf{A}_{l+1} \rightarrow \mathbf{B}_{m+1}) \wedge \phi(P) \\ &\quad \text{donde } l = \text{maxin}(\mathbf{A}, \phi(P)) \\ &\quad \text{y } m = \text{maxin}(\mathbf{B}, \phi(P)) \\ \phi(\mathbf{A} \not\Rightarrow \mathbf{B} \text{ in } P) &= (\neg \mathbf{A}_{l+1} \rightarrow \neg \mathbf{A}_l) \wedge (\neg \mathbf{B}_{m+1} \rightarrow \neg \mathbf{B}_m) \wedge \\ &\quad (\neg \mathbf{A}_{l+1} \vee \neg \mathbf{B}_{m+1}) \wedge \phi(P) \\ &\quad \text{donde } l = \text{maxin}(\mathbf{A}, \phi(P)) \\ &\quad \text{y } m = \text{maxin}(\mathbf{B}, \phi(P)) \\ \phi(P \Rightarrow \mathbf{A}) &= \mathbf{A}_{l+1} \wedge \phi(P) \\ &\quad \text{donde } l = \text{máx}(0, \text{maxin}(\mathbf{A}, \phi(P))) \\ \phi(P \setminus \mathbf{A}) &= \neg \mathbf{A}_l \wedge \phi(P) \\ &\quad \text{donde } l = \text{maxin}(\mathbf{A}, \phi(P)) \end{aligned}$$

□

Antes de mostrar el resultado que permita comprobar la factibilidad de cualquier término $P \in \text{SPLA}$, es necesario describir una propiedad preliminar de $\phi(P)$.

Lema 3.8.6 Dado un término $P \in \text{SPLA}$, una característica $\mathbf{A} \in \mathcal{F}$ y una evaluación v , tal que $v \models \phi(P)$ y $l \in \mathbb{N}$, tal que $v(\mathbf{A}_l) = 0$ y $\mathbf{A}_l \in \text{vars}(\phi(P))$, entonces $v(\mathbf{A}_k) = 0$ para $k \leq l$.

Demostración: La demostración será inmediata por la definición estructural de P .

- $P = \text{nil}$ o $P = \checkmark$: Es trivial ya que $\text{vars}(\phi(P)) = \emptyset$.
- $P = B; P'$: En este caso, $\phi(P) = B_{m+1} \wedge \phi(P')$ donde $m = \text{maxin}(B, \phi(P))$. Así que $A_l \neq B_{m+1}$, entonces $A \neq B$ o $A = B$ y $l \neq m + 1$. En el segundo caso, se puede deducir que $l \leq m$ por la definición de maxin . Entonces, en ambos casos $A_l \in \text{vars}(\phi(P'))$ y se obtiene el resultado por inducción estructural.
- $P = \bar{B}; P'$: Es trivial, ya que $\text{vars}(\phi(P)) = \emptyset$.
- $P = P_1 \vee P_2$: En este caso, $\phi(P) = \text{comp}(\phi(P_1))\phi(P_2) \vee \text{comp}(\phi(P_1))\phi(P_2)$. Ya que $v \models \phi(P)$, entonces $v \models \text{comp}(\phi(P_1))\phi(P_2)$ o $v \models \text{comp}(\phi(P_2))\phi(P_1)$. Asumiendo el primer caso, ya que $v \models \text{comp}(\phi(P_1))\phi(P_2)$ y $v \models \phi(P_1)$. De esta manera, se obtienen los siguientes casos
 - $A_l \in \text{vars}(\phi(P_1))$: Entonces, se obtiene el resultado por inducción sobre P_1 .
 - $A_l \notin \text{vars}(\phi(P_1))$: Considérese $m = \text{maxin}(A, \phi(P_1))$. Entonces, utilizando el lema 3.8.4.3, se obtiene que $m < l \leq \text{maxin}(A, \phi(P_2))$. Desde que, $v(A_l) = 0$ y $v \models \text{comp}(\phi(P_2))\phi(P_1)$ por la construcción de $\text{comp}(\phi(P_2))\phi(P_1)$, se obtiene que $v(A_k) = 0$ para $m \leq k \leq l$. Por lo tanto, $v(A_m) = 0$ y entonces, por inducción sobre P_1 se obtiene el resultado.
- $P = P_1 \wedge P_2$: Este caso, es realizado directamente por inducción estructural.
- $P = B \Rightarrow C$ in P' o $P = B \nRightarrow C$ in P' : Si $A \neq B$ o $A \neq C$ el resultado es obtenido al aplicar directamente inducción estructural. Asumiendo que $A = B$ (el caso cuando $A = C$ es idéntico al intercambiar B por C). Considerando que $m = \text{maxin}(B, \phi(P'))$. Si $l \leq m$, entonces $A_l \in \text{vars}(\phi(P'))$ y se obtiene el resultado por inducción estructural. Entonces, queda el caso cuando $l = m + 1$. Si $l = 0$ no hay nada que demostrar, así que se asume $l > 0$. Desde que $v \models \phi(P)$, se obtiene $v \models \neg A_{m+1} \rightarrow A_m$. Desde que $v(A_{m+1}) = 0$, entonces $m = -1$ o $v(A_m) = 0$, se asume $m \geq 0$ porque si $m = -1$ no

existe nada que demostrar. Por la definición de maxin , $\mathbf{A}_m \in \text{vars}(\phi(P'))$. Entonces, por inducción estructural, $v(\mathbf{A}_k) = 0$ para $k \leq m = l - 1$ que completa el resultado.

- $P = P' \Rightarrow B$: Este caso, es el mismo que $P = A; P$.
- $P = P' \setminus B$: Este caso, es realizado directamente por inducción estructural.

□

Se quiere demostrar que existe $p \in \text{SPLA}$ si y solo si la fórmula $\phi(P)$ es factible. Sin embargo, existen diversos aspectos a tener en cuenta con la presencia de restricciones, tales como las restricciones de requerimiento, exclusión, obligatoriedad o prohibición dentro de un operador de conjunción. De esta forma, el resultado está restringido a aquellos términos que no contengan el operador de conjunción. Es importante destacar que no representa una limitación, ya que las restricciones pueden ser consideradas externas a los demás operadores.

Definición 3.8.7 Dado un término $P \in \text{SPLA}$, se dice que P es una *SPL segura* si no existen restricciones dentro del operador de conjunción (\wedge). □

Proposición 3.8.8 Dada una SPL segura $P \in \text{SPLA}$, si $p \in \text{prod}(P)$ entonces existe una evaluación v tal que $v \models \phi(P)$ y $A \in p$, si y solo si $k \geq 0$ y $v(\mathbf{A}_k) = 1$, donde $k = \text{maxin}(A, \phi(P))$.

Demostración: La demostración será realizada por inducción estructural sobre P . Para todos los casos, será utilizado el teorema 3.4.12 ($\text{prod}(P) = \llbracket P \rrbracket$).

- $P = \text{nil}$ o $P = \checkmark$: Estos son los casos base y la demostración es inmediata. $\phi(\text{nil})$ no es factible $\text{prod}(\text{nil}) = \emptyset$. Mientras que la evaluación v tal que $v(\mathbf{A}_k) = 0$ para cualquier $A \in \mathcal{F}$ y $k \in \mathbb{N}$ cumple la tesis para el caso $P = \checkmark$.
- $P = A; P'$: En este caso, $\phi(A; P') = A_{l+1} \wedge \phi(P')$ donde $l = \text{maxin}(A, \phi(P'))$. Considérese $p \in \text{prod}(P)$, entonces existe $p' \in \text{prod}(P')$, tal que $p = \{A\} \cup p'$. Al aplicar inducción sobre P' existe una evaluación v' tal que se cumple la tesis para p' y P' .

Considérese la evaluación $v = v'[A_{l+1}/1]$ ⁴. No es difícil comprobar que v mantiene la tesis para p y P :

- $v \models A_{l+1} \wedge \phi(P')$ desde que $A_{l+1} \notin \text{vars}(\phi(P'))$.
 - Considérese $B \in p$. Entonces $B = A$ o $B \in p'$. En el primer caso $v(A_{l+1}) = 1$. En el segundo caso, por inducción se obtiene que, $k \geq 0$ y $v'(B_k) = 1$ donde $k = \text{maxin}(B, \phi(P'))$. Entonces $v'(B_k) = 1$ y $k = \text{maxin}(B, \phi(P)) = \text{maxin}(B, \phi(P'))$.
 - Considérese $B \notin p$. Entonces $B \neq A$ y $B \notin p'$. Por inducción, $k = -1$ o $v'(B_k) = 0$, donde $k = \text{maxin}(B, \text{vars}\phi(P'))$. Entonces $k = \text{maxin}(B, \text{vars}\phi(P))$ y $k < 0$ o $v(B_k) = 0$.
- **$P = P_1 \vee P_2$:** En este caso $\text{prod}(P) = \text{prod}(P_1) \cup \text{prod}(P_2)$. Por lo tanto $p \in \text{prod}(P)$ si y solo si $p \in \text{prod}(P_1)$ o $p \in \text{prod}(P_2)$. Suponemos que $p \in \text{prod}(P_1)$ (el otro caso es simétrico). Por inducción, existe una evaluación v que satisface la tesis para p y P_1 . Ahora, consideramos la evaluación v' definida en la demostración del lema 3.8.4.2. A continuación será demostrado que v' cumple la tesis.
- Desde que $v \models P_1$, entonces $v' \models \text{comp}(\phi(P_1))\phi(P_2)$.
 - Ahora, considérese una característica $A \in p$. Por inducción $v(A_k) = 1$, donde $k = \text{maxin}(A, \phi(P_1))$. Entonces $v'(A_k) = 1$ por la construcción de v' . Si $k = \text{maxin}(A, \phi(P))$ se obtiene el resultado. Así que, supongamos que $k < \text{maxin}(A, \phi(P))$, dado $m = \text{maxin}(A, \phi(P))$. Desde que $v' \models \phi(P)$, si $v(A_m) = 0$, por el lema 3.8.6, se puede concluir que $v(A_k) = 0$, por lo tanto $v(A_m) = 1$.
 - Finalmente, considérese $A \notin p$. Por inducción $m = -1$ o $v(A_m) = 0$ donde $m = \text{maxin}(A, \phi(P_1))$. Si $m = \text{maxin}(A, \phi(P))$, por la construcción de v' , $v'(A_m) = 0$. Si $m < \text{maxin}(A, \phi(P))$, entonces considérese $l = \text{maxin}(A, \phi(P_2)) = \text{maxin}(A, \text{comp}(\phi(P_1))\phi(P_2))$. Por la construcción de v' , $v'(A_l) = v(A_m) = 0$.

⁴ $v[A/x](B) = v(B)$ si $A \neq B$ y $v[A/x](A) = x$

- $P = \bar{A}; P'$: Este es un caso particular del caso anterior desde que $P =_E \checkmark \vee A; P'$.
- $P = P_1 \wedge P_2$: En este caso, $p \in \text{prod}(P)$ si y solo si existe $p_1 \in \text{prod}(P_1)$ y $p_2 \in \text{prod}(P_2)$ tal que $p = p_1 \cup p_2$. Por inducción de P_1 y P_2 , existe una evaluación v_1 manteniendo la tesis para p_1 y P_1 , y v_2 manteniendo la tesis para p_2 y P_2 . Considérese la evaluación v definida como:

$$v(x) = \begin{cases} 1 & \text{si } v_1(x) = 1 \text{ y } x \in \text{vars}(\phi(P_1)) \\ 1 & \text{si } v_2(x) = 1 \text{ y } x \in \text{vars}(\phi(P_2)) \\ 0 & \text{en otro caso} \end{cases}$$

Esta evaluación mantiene la tesis para p y P :

- Desde que P es seguro y el único operador que introduce variables lógicas negadas son las restricciones, no existe ninguna variable lógica negada en $\phi(P_1)$ o en $\phi(P_2)$. Por lo tanto $v \models \phi(P) = \phi(P_1) \wedge \phi(P_2)$.
- Considérese una característica $A \in p$. Entonces, $A \in p_1$ o $A \in p_2$. Asumiendo que $A \in p_1$, el otro caso es simétrico. De esta manera, por inducción, $v_1(A_l) = 1$ donde $l = \text{maxin}(A, \phi(P_1))$. Así, por la construcción de v , $v(A_l) = 1$. Considérese $m = \text{maxin}(A, \phi(P))$. Si $v(A_m) = 0$, utilizando el lema 3.8.6 $v(A_l) = 0$, entonces $v(A_m) = 1$.
- Considérese $A \notin p$. Entonces $A \notin p_1$ y $A \notin p_2$. Considérese también que $l = \text{maxin}(A, \phi(P))$. Entonces $l = \text{maxin}(A, \phi(P_1))$ o $l = \text{maxin}(A, \phi(P_2))$. Suponiendo que $l = \text{maxin}(A, \phi(P_1))$, el otro caso es simétrico. Desde que $A \notin p_1$, $l = -1$ o $v_1(A) = 0$. Existen dos posibilidades, que $l = \text{maxin}(A, \phi(P_2))$ o que $l < \text{maxin}(A, \phi(P_2))$. En el segundo caso $A_l \notin \text{vars}(\phi(P_2))$, entonces por la construcción de v , $v(A_l) = 0$. En el primer caso, por inducción $l = -1$ o $v_2(A_l) = 0$, entonces por la construcción de v , $v(A_l) = 0$.
- $P = A \Rightarrow B$ in P' : En este caso, $\phi(P) = (\neg A_{l+1} \rightarrow \neg A_l) \wedge (\neg B_{m+1} \rightarrow \neg B_m) \wedge (A_{l+1} \rightarrow B_{m+1}) \wedge \phi(P')$, donde $l = \text{maxin}(A, \phi(P'))$ y $m = \text{maxin}(B, \phi(P'))$.

Considerando el producto $p \in \text{prod}(P)$, si y solo si existe un producto $p' \in \text{prod}(P')$, tal que $p = p'$ y $A \notin p'$ o $p = p' \cup \{B\}$ y $A \in p'$. En ambos casos, por inducción, existe una evaluación v' que satisface la tesis para p' y para P' .

Considérese la siguiente evaluación v acorde a los siguientes casos:

- $A \in p'$: Entonces $v = v'[A_{l+1}/1, B_{m+1}/1]$.
- $A \notin p'$ y $B \in p'$: Entonces $v = v'[A_{l+1}/0, B_{m+1}/1]$.
- $A \notin p'$ y $B \notin p'$: Entonces $v = v'[A_{l+1}/0, B_{m+1}/0]$.

No es difícil comprobar que los tres casos v satisfacen las condiciones para p y P :

- Debido a la manera en que v y $\phi(P)$ están definidas, $v \models \phi(P)$.
- Considérese $C \in p$. Si $C = B$, existen dos casos $B \in p'$ o $A \in p'$. En ambos casos, por la construcción de v , $v(C_{m+1}) = 1$. Si $C = A$, entonces $A \in p'$, así que $v(A_{l+1}) = 1$. Si $C \neq B$ y $C \neq A$, entonces $C \in p'$ y por inducción $l \geq 0$ y $v'(C_l) = v(C_l) = 1$ para $l = \text{maxin}(C, \phi(P')) = \text{maxin}(C, \phi(P))$.
- Considérese $C \notin p$. Entonces $C \notin p'$. Por inducción $l < 0$ o $v'(C_l) = 0$ para $l = \text{maxin}(C, \phi(P'))$. Si $C \neq A$ o $C \neq B$, entonces $l = \text{maxin}(C, \phi(P'))$ y $l < 0$ o $v(C_l) = v'(C_l) = 0$. Si $C = B$, entonces $B \notin p'$ y, por la construcción de v , $v(B_{m+1}) = 0$. Si $C = A$, entonces $A \notin p'$ y, por la construcción de v , $v(A_{l+1}) = 0$.
- $P = A \not\rightarrow B$ in P' : En este caso $\phi(P) = (\neg A_{l+1} \rightarrow \neg A_l) \wedge (\neg B_{m+1} \rightarrow \neg B_m) \wedge (\neg A_{l+1} \vee \neg B_{m+1}) \wedge \phi(P')$, donde $l = \text{maxin}(A, \phi(P'))$ y $m = \text{maxin}(B, \phi(P))$. Mas allá, $p \in \text{prod}(P)$ si y solo si $p \in \text{prod}(P')$ y $A \notin p$ o $B \notin p$. Asumiendo el primer caso (el otro es simétrico). Por inducción de la hipótesis, existe una evaluación v' que satisface la tesis para p y P' . Ahora hay dos casos, $B \in p$ o $B \notin p$, en el primer caso considérese que $v = v'[A_{l+1}/0, B_{m+1}/1]$, en el segundo caso considérese que $v = v'[A_{l+1}/0, B_{m+1}/0]$. Ahora es necesario comprobar que v satisface la tesis para p y P :

- Es necesario comprobar que $v \models \phi(P)$ al analizar sus partes. Desde que $A \notin p$, por inducción $l < 0$ o $v(A_l) = 0$, entonces $v \models \neg A_{l+1} \rightarrow \neg A_l$. De igual manera por inducción, $B \in p$ si y solo si $B \in p'$ si y solo si $m \geq 0$ y $v'(B_m) = 1$. Por lo tanto, $v \models \neg B_{m+1} \rightarrow \neg B_m$. Por la construcción de $v \models \neg A_{l+1} \vee \neg B_{m+1}$ y por inducción $v' \models \phi(P')$. Desde que $A_{l+1}, B_{m+1} \notin \text{vars}(\phi(P'))$, $v \models \phi(P')$.
 - Considérese que $C \in p$, desde que está considerado que $A \notin p'$, $C \neq A$. Si $C = B$, entonces $B \in p'$. Por lo tanto, por la construcción de v , $v(B_{m+1}) = 1$. En caso contrario por la hipótesis de inducción, $k \geq 0$ y $v'(C_k) = 1$ para $k = \text{maxin}(C, \phi(P'))$. Entonces $k = \text{maxin}(C, \phi(P))$ y $v(C_k) = v'(C_k) = 1$.
 - Considérese $C \notin p$. Si $C = A$ entonces $v(A_{l+1}) = 0$. Si $C = B$ entonces $v(B_{m+1}) = 0$. Si $C \neq A$ y $C \neq B$, por inducción $k < 0$ o $v'(C_k) = 0$ para $k = \text{maxin}(C, \phi(P'))$. Debe observarse que $k = \text{maxin}(C, \phi(P))$ y $v(C_k) = v'(C_k) = 0$.
- $P = P' \Rightarrow A$: Este caso es el mismo que $P = A; P$.
- $P = P' \setminus A$: En este caso, $\phi(P) = \neg A_l \wedge \phi(P)$ donde $l = \text{maxin}(A, \phi(P'))$. Considérese el término $p \in \text{prod}(P)$, entonces existe un producto $p \in \text{prod}(P')$ tal que $A \notin p$. Por inducción estructural, existe una evaluación v que sostiene la tesis para p y P' . Es necesario demostrar que v también satisface las condiciones para P .
- Primero $v \models \phi(P)$. Existen dos casos: $l = -1$ o $l \geq 0$. En el primer caso, A_{l-1} es el símbolo \perp . En el segundo caso, desde que $A \notin p$, por inducción $v(A_l) = 0$. En cualquier caso, por la hipótesis de inducción, $v \models \neg A_l \wedge \phi(P')$.
 - Considérese una característica B . Por inducción, $B \in p$ si y solo si $l > 0$ y $v(B_l) = 1$ donde $l = \text{maxin}(B, \phi(P'))$. Para concluir este punto es suficiente con considerar que $\text{vars}(\phi(P)) = \text{vars}(\phi(P'))$.

La segunda condición se sostiene trivialmente desde que $\text{vars}(\phi(P)) = \text{vars}(\phi(P'))$.

□

La proposición 3.8.9 describe la *implicación de derecha a izquierda* del teorema 3.8.10. También se demuestra por inducción estructural sobre P .

Proposición 3.8.9 Dada una SPL segura $P \in \text{SPLA}$, si existe una evaluación v tal que $v \models \phi(P)$, entonces existe un producto $p \in \text{prod}(P)$, tal que $\mathbf{A} \notin p$ para cualquier característica \mathbf{A} , satisfaciendo $v(\mathbf{A}_l) = 0$ para todo $0 \leq l \leq \text{maxin}(\mathbf{A}, \phi(P))$.

Demostración: La demostración es realizada por inducción estructural sobre P .

- $P = \text{nil}$ o $P = \checkmark$: Estos son los casos base, y su demostración es inmediata. $\phi(\text{nil})$ no es factible y $\text{prod}(\text{nil}) = \emptyset$. Mientras que el único producto de \checkmark , que es el conjunto vacío \emptyset no tiene características.
- $P = \mathbf{A}; P'$: En este caso $\phi(\mathbf{A}; P') = \mathbf{A}_{l+1} \wedge \phi(P')$ donde $l = \text{máx}(0, \text{maxin}(\mathbf{A}, \phi(P')))$. Si $v \models \phi(P)$, entonces $v \models \phi(P')$. Por inducción estructural, existe un producto $p' \in \text{prod}(P')$ satisfaciendo la tesis. Considérese $p = p' \cup \{\mathbf{A}\} \in \text{prod}(P)$. Por la definición de la semántica denotacional del operador de prefijo, $p \in \text{prod}(\mathbf{A}; P')$.

Ahora considérese una característica \mathbf{C} , tal que $v(\mathbf{C}_k) = 0$ para cualquier $0 \leq k \leq \text{maxin}(\mathbf{C}, \phi(P))$, desde que $v \models \phi(P)$, $v(\mathbf{A}_{l+1}) = 0$, $\mathbf{C} \neq \mathbf{A}$. Por inducción, $\mathbf{C} \notin p'$, entonces por la construcción de p , $\mathbf{C} \notin p$.

- $P = P_1 \vee P_2$: En este caso, $\phi P = \text{comp}(\phi(P_1))\phi(P_2) \vee \text{comp}(\phi(P_2))\phi(P_1)$. Desde que $v \models \phi(P)$, entonces $v \models \text{comp}(\phi(P_1))\phi(P_2)$ o $v \models \text{comp}(\phi(P_2))\phi(P_1)$. Suponiendo que $v \models \text{comp}(\phi(P_2))\phi(P_1)$ (el otro caso es simétrico). Por el lema 3.8.4.1 $v \models \phi P_2$. Entonces, por inducción, existe un producto $p \in \text{prod}(P_2)$ que cumple la tesis para p y P_2 . Desde que $\text{prod}(P) = \text{prod}(P_1) \cup \text{prod}(P_2)$, entonces $p \in \text{prod}(P)$. Ahora considérese una característica \mathbf{A} , tal que $v(\mathbf{A}_l) = 0$ para cualquier $0 \leq l \leq \text{maxin}(\mathbf{A}, \phi(P))$. Desde que $\text{maxin}(\mathbf{A}, \phi(P)) \geq \text{maxin}(\mathbf{A}, \phi(P_2))$, por inducción, $\mathbf{A} \notin p$.
- $P = \bar{\mathbf{A}}; P'$: Este caso es un caso particular del caso anterior desde que $P =_E \checkmark \vee \mathbf{A}; P'$.

- **$P = P_1 \wedge P_2$:** En este caso $\phi(P) = \phi(P_1) \wedge \phi(P_2)$. Desde que $v \models \phi(P)$, entonces $v \models \phi(P_1)$ y $v \models \phi(P_2)$. Por inducción estructural, existe un $p_1 \in \text{prod}(P_1)$ que cumple la tesis para v y P_1 y existe un $p_2 \in \text{prod}(P_2)$ que cumple la tesis para v y P_2 . Al mostrar $p = p_1 \cup p_2$ se cumple la tesis para v y P . Primero, por la definición de los operadores $p \in \text{prod}(P)$. Considérese una característica A tal que $v(A_l) = l$ para todos $0 \leq k \leq \text{maxin}(A, \phi(P))$. Desde que $\text{maxin}(A, \phi(P)) \geq \text{maxin}(A, \phi(P_1))$ y $\text{maxin}(A, \phi(P)) \geq \text{maxin}(A, \phi(P_2))$, por inducción, $A \notin p_1$ y $A \notin p_2$. Por lo tanto, $A \notin p_1 \cup p_2 = p$.
- **$P = A \Rightarrow B$ in P' :** En este caso, $\phi(P) = (\neg A_{l+1} \rightarrow \neg A_l) \wedge (\neg B_{m+1} \rightarrow \neg B_m) \wedge (A_{l+1} \rightarrow B_{m+1}) \wedge \phi(P')$ donde $l = \text{maxin}(A, \phi(P'))$ y $m = \text{maxin}(B, \phi(P'))$. Considérese v una evaluación tal que $v \models \phi(P)$. Por inducción, existe un $p' \in \text{prod}(P')$ que mantiene la tesis para v y P .

Existen dos casos $A \in p'$ o $A \notin p'$. En el primer caso, considérese que $p = p' \cup \{B\} \in \text{prod}(P)$ y en el segundo caso $p = p'$. Comprobando que en ambos casos, p mantiene la tesis para v y P . Por la definición de la semántica denotacional, $p \in \text{prod}(P)$. Ahora considérese la característica C , tal que $v(C_k) = 0$ para todos con $k \in \mathbb{N}$. Existen los siguientes casos:

- **$C = A$:** Por inducción $A \notin p'$. Entonces, por la construcción de p , $A \notin p$.
- **$C = B$:** Por inducción $B \notin p'$. Desde que $v(B_{m+1}) = 0$ y $v \models \phi(P)$, $v(A_{l+1}) = 0$. Entonces, $l = -1$ or $v(A_l) = 0$. Luego, por el lema 3.8.6, $v(A_k) = 0$ para todos $0 \leq k < l$. Entonces, por inducción, $A \notin p'$. Desde que $B \notin p'$ y $A \notin p'$, por la construcción de p , $B \notin p$.
- **$C \neq B$ y $C \neq A$:** En este caso, por inducción, $C \notin p'$. Desde que $C \neq B$ y $C \neq A$, por la construcción de p , $C \notin p$.
- **$P = A \not\Rightarrow B$ in P' :** En este caso $\phi(P) = (\neg A_{l+1} \rightarrow \neg A_l) \wedge (\neg B_{m+1} \rightarrow \neg B_m) \wedge (\neg A_{l+1} \vee \neg B_{m+1}) \wedge \phi(P')$ donde $l = \text{maxin}(A, \phi(P'))$ y $m = \text{maxin}(B, \phi(P'))$.

Considérese una evaluación v , tal que $v \models \phi(P)$. Entonces $v \models \phi(P')$, y por inducción, existe un producto $p \in \mathbf{prod}(P')$ que sostiene la tesis para v y P' . Ahora, hay que mostrar que p también sostiene la tesis para v y P . Primero, debe ser demostrado que $p \in \mathbf{prod}(P)$. Desde que $v \models \phi(P)$, $v(A_{l+1}) = 0$ o $v(B_{m+1}) = 0$, asumiendo que $v(B_{m+1}) = 0$, el otro caso es simétrico. Desde que $v \models \phi(P)$, $m = -1$ o $v(B_m) = 0$, entonces por el lema 3.8.6, $v(B_k) = 0$ para $0 \leq k \leq m$. De esta manera, por inducción, $B \notin p$. Así que, por la definición de la semántica denotacional, $p \in \mathbf{prod}(P)$.

Ahora, considérese una característica \mathbf{C} tal que $v(\mathbf{C}_k) = 0$ para todos $0 \leq k \leq \mathbf{maxin}(\mathbf{C}, \phi(P))$. Existen los siguientes casos:

- $\mathbf{C} = \mathbf{A}$: Entonces $k = l + 1$ y $l = \mathbf{maxin}(\mathbf{A}, \phi(P'))$. Desde que $v \models \phi(P)$ y $v(A_{l+1}) = 0$, $l = -1$ o $v(A_l) = 0$. Así, por el lema 3.8.6, $v(A_k) = 0$ para $0 \leq k \leq l$. Por lo tanto, por inducción, $\mathbf{A} \notin p$.
 - $\mathbf{C} = \mathbf{B}$: Desde que $v \models \phi(P)$ y $v(B_{m+1}) = 0$, $m = -1$ o $v(B_m) = 0$. Así por el lema 3.8.6, $v(B_k) = 0$ para $0 \leq k \leq m$. Entonces, por inducción $\mathbf{B} \notin p$.
 - $\mathbf{C} \neq \mathbf{B}$ y $\mathbf{C} \neq \mathbf{A}$: En este caso es suficiente considerar que $\mathbf{maxin}(\mathbf{C}, \phi(P')) = \mathbf{maxin}(\mathbf{C}, \phi(P))$. Entonces, por inducción, $\mathbf{C} \notin p$.
- $P = P' \Rightarrow \mathbf{A}$: Este caso es el mismo que $P = \mathbf{A}; P$.
- $P = P' \setminus \mathbf{A}$: En este caso $\phi(P) = \neg A_l \wedge \phi(P')$ donde $l = \mathbf{maxin}(\mathbf{A}, \phi(P'))$. Considerando la evaluación v , tal que $v \models \phi(P)$. Entonces $v \models \phi(P')$. Por inducción, existe un producto $p \in \mathbf{prod}(P')$ sosteniendo la tesis para v y P' . Es necesario comprobar que p sostiene la tesis para v y P . Primero, es necesario mostrar que $\mathbf{A} \notin p$. Desde que $v \models \phi(P)$, existen dos casos $l = -1$ o $l \geq 0$ y $v(A_l) = 0$. Así, por el lema 3.8.6, $v(A_k) = 0$ para $0 \leq k \leq l$. Entonces, por inducción, $\mathbf{A} \notin p$. De esta manera, por la definición del operador semántico $\llbracket \cdot \setminus \mathbf{A} \rrbracket$, $p \in \mathbf{prod}(P)$. Ahora, considérese una característica \mathbf{C} tal que $v(\mathbf{C}_k) = 0$ para $0 \leq k \leq \mathbf{maxin}(\mathbf{C}, \phi(P))$. Ya ha sido demostrado que $\mathbf{A} \notin p$, así que

puede asumirse que $\mathbf{C} \neq \mathbf{A}$. Desde que $\text{maxin}(\mathbf{C}, \phi(P)) = \text{maxin}(\mathbf{C}, \phi(P'))$, se obtiene $\mathbf{C} \notin P$ por inducción.

□

De esta forma se obtiene el resultado buscado.

Teorema 3.8.10 Dada una SPL segura $P \in \text{SPLA}$, entonces $p \in \text{prod}(P)$ si y solo si existe una evaluación v tal que $v \models \phi(P)$. □

El resultado principal de esta sección es el teorema 3.8.10, el cual, está relacionado con la factibilidad de una SPL P y de la factibilidad de su fórmula asociada $\phi(P)$. Este teorema es una consecuencia directa de la proposición 3.8.8 y de la proposición 3.8.9. La proposición 3.8.8 es la *implicación de izquierda de derecha* del teorema 3.8.10, el cual ha sido demostrado por inducción estructural de P , donde la proposición 3.8.8 y la proposición 3.8.9 son necesarias para demostrar el resultado.

Los resultados teóricos de esta sección son descritos en la sección 3.10, donde se presenta una herramienta que, haciendo uso de *SAT solvers*, determina si una SPL P genera productos válidos o no. Es importante destacar que $\phi(P)$ no es una fórmula en forma normal conjuntiva (CNF, del inglés Conjunctive Normal Form).

Por ello, para comprobar su factibilidad, $\phi(P)$ debe ser convertido a una expresión CNF.

3.9. Caso de estudio: Sistema VSS

En esta sección se modela una herramienta de *streaming* de video. El diagrama FODA del sistema de streaming está descrito en la figura 3.12. Este sistema incorpora las siguientes características: VSS, TBR, VCC, 720Kbps, 256Kbps, H.264 y MPEG.2. La *característica inicial*⁵ para esta SPL es el software de streaming de video (VSS, del inglés Video Streaming

⁵Llamado de manera global como la SPL.

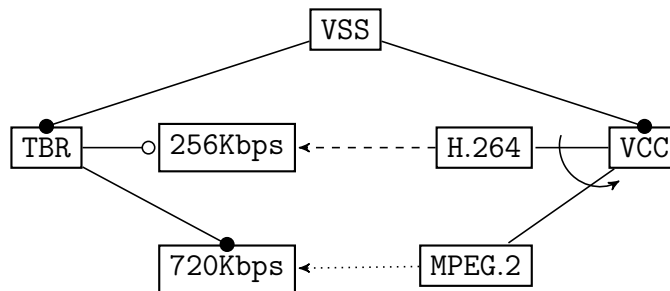


Figura 3.12: Representación FODA del *software* de *streaming* de video.

Software). Cualquier producto de esta SPL necesitará esta característica principal. Es importante destacar que cada característica tiene un nombre único dentro del diccionario de características de VSS.

Las características relacionadas con la característica inicial VSS, son la tasa de transmisión (TBR, del inglés transmission bitrate) y el codec de video (VCC, del inglés video codec), las cuales están relacionadas entre si de manera *obligatoria*. Esta relación establece que este conjunto de características estará incluido en todas las implementaciones del software donde VSS esté incluida.

Además, existe otra relación de *obligatoriedad* en el diagrama, la característica 720Kbps con la característica del nivel superior. De igual, manera existen otras características opcionales en el sistema. Por ejemplo, la característica 256Kbps es opcional, lo que significa que puede, o no, incluirse en los productos finales de la SPL. Es importante destacar que la inclusión de esta característica depende de otras relaciones representadas como restricciones. Finalmente, las características H.264 y MPEG.2 están relacionadas con el operador de *selección única*, lo que significa que al menos una de las dos características estará incluida en los productos finales de la SPL.

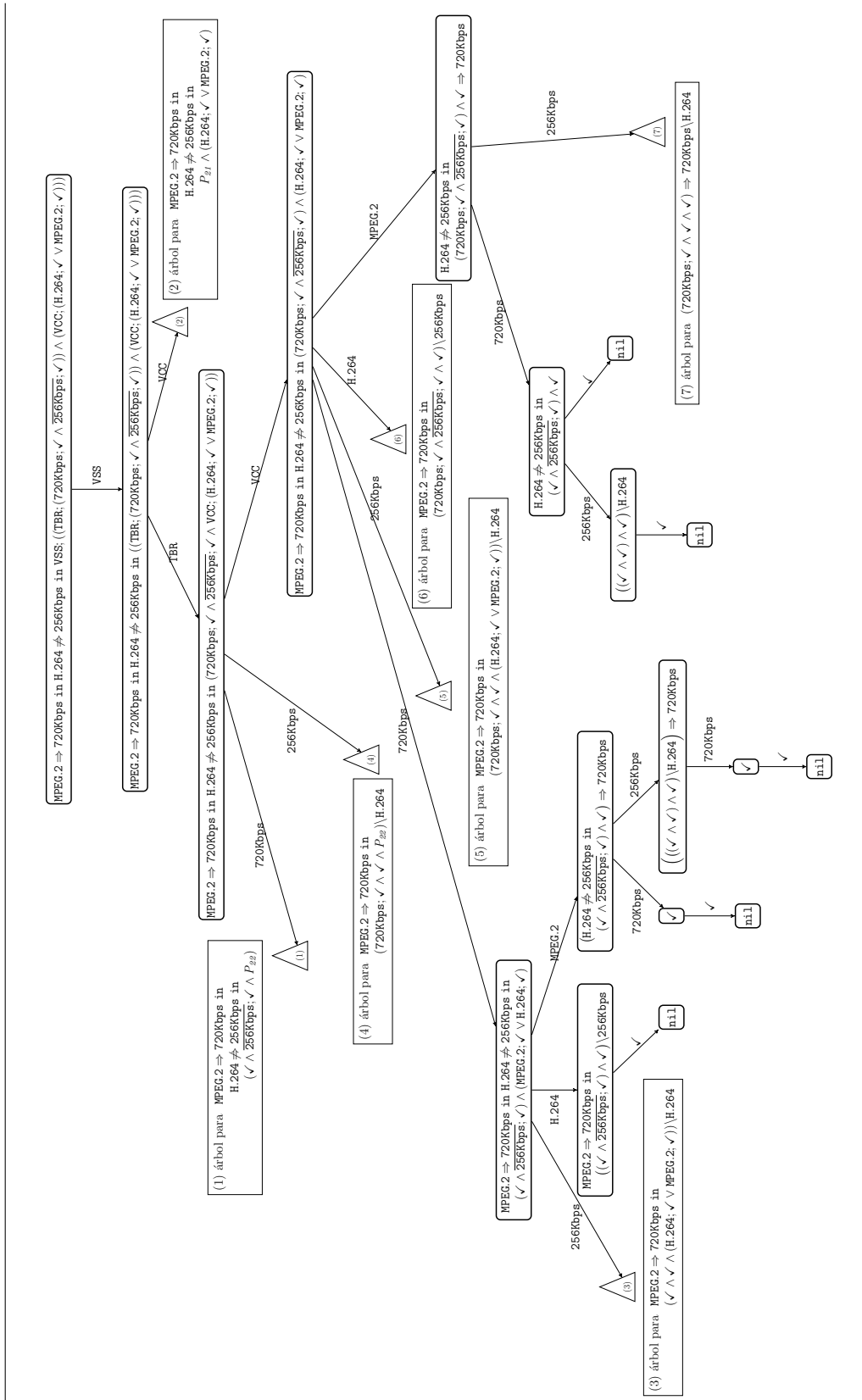


Figura 3.13: Sistema de etiquetado de transiciones para VSS.

En este sistema existe un conjunto de *reglas* que describen un grupo de *restricciones*. Por ejemplo, cuando la característica MPEG.2 está incluida en un producto, la característica 720Kbps también debe estarlo. De forma similar, cuando la característica H.264 es seleccionada, 256Kbps no debe aparecer. La restricción que relaciona MPEG.2 con la velocidad 720Kbps es una restricción de *requerimiento*, lo que significa si MPEG.2 es seleccionada, entonces la velocidad TBR debe ser 720Kbps.

Existe otra restricción de *exclusión* que relaciona las características H.264 y 256Kbps, esto es si H.264 aparece en el producto final, entonces éste no puede contener 256Kbps.

Finalmente, un producto *válido* del modelo es el conjunto de características que satisfacen todas las restricciones del diagrama.

Por ejemplo, consideramos los siguientes productos, pr_1 compuesto por VSS, TBR, 720Kbps, VCC, y H.264, pr_2 compuesto por VSS, TBR, 720Kbps, VCC, y MPEG.2 y pr_3 compuesto por VSS, TBR, 720Kbps, 256Kbps, VCC, y H.264. En este caso, pr_1 y pr_2 son productos *válidos* del modelo, mientras que pr_3 no representa un producto válido.

Para mostrar el análisis formal, esta fase se centrará en describir el *proceso de traducción*, donde la traducción completa de este modelo está representada como P_1 :

$$\begin{aligned} P_1 &:= \text{MPEG.2} \Rightarrow 720\text{Kbps in } (\text{H.264} \not\Rightarrow 256\text{Kbps in } P_2) \\ P_2 &:= \text{VSS}; (P_{21} \wedge P_{22}) \\ P_{21} &:= \text{TBR}; (720\text{Kbps}; \checkmark \wedge \overline{256\text{Kbps}}; \checkmark) \\ P_{22} &:= \text{VCC}; (\text{H.264}; \checkmark \vee \text{MPEG.2}; \checkmark) \end{aligned}$$

La figura 3.13 muestra el sistema de etiquetado de transiciones asociado a la semántica operacional de SPLA para el término P_1 . En esta figura se han omitido los sub-árboles del (1) al (7) ya que no es posible representarlos en su descripción gráfica.

En el árbol descrito se obtienen las siguientes trazas:

- VSS TBR VCC 720Kbps H.264
- VSS TBR VCC 720Kbps MPEG.2 720Kbps
- VSS TBR VCC 720Kbps MPEG.2 256Kbps 720Kbps

- VSS TBR VCC MPEG.2 720Kbps
- VSS TBR VCC MPEG.2 720Kbps 256Kbps

De estas trazas serán obtenidos los siguientes productos:

- {VSS, TBR, VCC, 720Kbps, H.264}
- {VSS, TBR, VCC, 720Kbps, MPEG.2}
- {VSS, TBR, VCC, 720Kbps, MPEG.2, 256Kbps}

$$\begin{aligned}
& \llbracket \begin{array}{l} \checkmark \\ \text{H.264; } \checkmark \\ \text{MPEG.2; } \checkmark \\ \text{720Kbps; } \checkmark \\ \text{256Kbps; } \checkmark \end{array} \rrbracket &= \begin{array}{l} \{\emptyset\} \\ \{\{\text{H.264}\} \cup \emptyset\} = \{\{\text{H.264}\}\} \\ \{\{\text{MPEG.2}\} \cup \emptyset\} = \{\{\text{MPEG.2}\}\} \\ \{\{\text{720Kbps}\} \cup \emptyset\} = \{\{\text{720Kbps}\}\} \\ \{\emptyset\} \cup \{\{\text{256Kbps}\} \cup \emptyset\} = \{\emptyset, \{\text{256Kbps}\}\} \end{array} \\
\\
& \llbracket P_{22} \rrbracket &= \llbracket \text{VCC; (H.264; } \checkmark \vee \text{MPEG.2; } \checkmark) \rrbracket = \llbracket \text{VCC; } \cdot \rrbracket (\llbracket \vee \rrbracket (\{\{\text{H.264}\}\}, \{\{\text{MPEG.2}\}\})) = \\
& \llbracket P_{21} \rrbracket &= \llbracket \text{VCC; } \cdot \rrbracket (\llbracket \text{H.264; } \checkmark \rrbracket \cup \llbracket \text{MPEG.2; } \checkmark \rrbracket) = \llbracket \text{VCC; } \cdot \rrbracket (\{\{\text{H.264}\}, \{\text{MPEG.2}\}\}) = \\
& & \{\{\text{VCC, H.264}\}, \{\text{VCC, MPEG.2}\}\} \\
& \llbracket P_{21} \rrbracket &= \llbracket \text{TBR; (720Kbps; } \checkmark \wedge \text{256Kbps; } \checkmark) \rrbracket = \\
& & \llbracket \text{TBR; } \cdot \rrbracket (\llbracket \wedge \rrbracket (\{\{\text{720Kbps}\}, \{\emptyset, \{\text{256Kbps}\}\}\})) = \\
& & \llbracket \text{TBR; } \cdot \rrbracket (\{\{\text{720Kbps}\}, \{\text{720Kbps, 256Kbps}\}\}) = \\
& & \{\{\text{TBR, 720Kbps}\}, \{\text{TBR, 720Kbps, 256Kbps}\}\} \\
\\
& \llbracket P_{21} \wedge P_{22} \rrbracket &= \llbracket \wedge \rrbracket \left(\begin{array}{l} \{\{\text{VCC, H.264}\}, \{\text{VCC, MPEG.2}\}\}, \{\{\text{TBR, 720Kbps}\}, \{\text{TBR, 720Kbps, 256Kbps}\}\} \\ \{\{\text{VCC, TBR, H.264, 720Kbps}\}, \{\text{VCC, TBR, H.264, 720Kbps, 256Kbps}\}\}, \\ \{\{\text{VCC, TBR, MPEG.2, 720Kbps}\}, \{\text{VCC, TBR, MPEG.2, 720Kbps, 256Kbps}\}\} \end{array} \right) \\
& \llbracket P_2 \rrbracket &= \llbracket \text{VSS; } (P_{21} \wedge P_{22}) \rrbracket = \llbracket \text{VSS; } \cdot \rrbracket (P_{21} \wedge P_{22}) = \\
& & \left\{ \begin{array}{l} \{\text{VSS, VCC, TBR, H.264, 720Kbps}\}, \{\text{VSS, VCC, TBR, H.264, 720Kbps, 256Kbps}\}, \\ \{\text{VSS, VCC, TBR, MPEG.2, 720Kbps}\}, \{\text{VSS, VCC, TBR, MPEG.2, 720Kbps, 256Kbps}\} \end{array} \right\} \\
\\
& \llbracket \text{H.264} \not\Rightarrow \text{256Kbps in } P_2 \rrbracket &= \llbracket \text{H.264} \not\Rightarrow \text{256Kbps in } \cdot \rrbracket (\llbracket P_2 \rrbracket) = \\
& & \left\{ \begin{array}{l} \{\text{VSS, VCC, TBR, H.264, 720Kbps}\}, \\ \{\text{VSS, VCC, TBR, MPEG.2, 720Kbps}\}, \\ \{\text{VSS, VCC, TBR, MPEG.2, 720Kbps, 256Kbps}\} \end{array} \right\} \\
& \llbracket \begin{array}{l} \text{MPEG.2} \Rightarrow \text{720Kbps in} \\ \text{H.264} \not\Rightarrow \text{256Kbps in } P_2 \end{array} \rrbracket &= \llbracket \text{MPEG.2} \Rightarrow \text{720Kbps in } \cdot \rrbracket (\llbracket \text{H.264} \not\Rightarrow \text{256Kbps in } P_2 \rrbracket) = \\
& & \left\{ \begin{array}{l} \{\text{VSS, VCC, TBR, H.264, 720Kbps}\}, \{\text{VSS, VCC, TBR, MPEG.2, 720Kbps}\}, \\ \{\text{VSS, VCC, TBR, MPEG.2, 720Kbps, 256Kbps}\} \end{array} \right\}
\end{aligned}$$

Figura 3.14: Semántica denotacional para VSS.

La semántica denotacional para P_I se presenta en la figura 3.14. Es importante destacar que el conjunto de productos no se ha modificado una vez aplicado el último operador semántico $\llbracket \text{MPEG.2} \Rightarrow \text{720Kbps in } \cdot \rrbracket$. Esto significa que la restricción de *requerimiento* no es

necesaria en este caso. Tal y como se esperaba, el conjunto de productos obtenidos al aplicar las reglas de la semántica denotacional coincide con el conjunto de productos obtenido de aplicar las reglas de la semántica operacional: $\text{prod}(P_I) = \llbracket P_I \rrbracket$ (ver teorema 3.4.12).

$$\begin{aligned}
& \text{TBR}; (720\text{Kbps}; \checkmark \wedge \overline{256\text{Kbps}}; \checkmark) =_E \quad [\text{CON1}] \\
& \text{TBR}; 720\text{Kbps}; (\checkmark \wedge \overline{256\text{Kbps}}; \checkmark) =_E \quad [\text{CON5}] \quad [\text{CON2}] \\
& \quad \text{TBR}; 720\text{Kbps}; \overline{256\text{Kbps}}; \checkmark =_E \quad [\text{PRE2}] \\
& \quad \text{TBR}; 720\text{Kbps}; (256\text{Kbps}; \checkmark \vee \checkmark)
\end{aligned} \tag{3.2}$$

$$\begin{aligned}
& P_2 = \text{VSS}; (P_{21} \wedge P_{22}) =_E \tag{3.2} \\
& \text{VSS}; (\text{TBR}; 720\text{Kbps}; (256\text{Kbps}; \checkmark \vee \checkmark)) \wedge (\text{VCC}; (\text{H.264}; \checkmark \vee \text{MPEG.2}; \checkmark)) =_E \quad [\text{CON1}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; ((256\text{Kbps}; \checkmark \vee \checkmark) \wedge (\text{H.264}; \checkmark \vee \text{MPEG.2}; \checkmark)) =_E \quad [\text{CON3}] \\
& \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (((256\text{Kbps}; \checkmark \vee \checkmark) \wedge \text{H.264}; \checkmark) \vee ((256\text{Kbps}; \checkmark \vee \checkmark) \wedge \text{MPEG.2}; \checkmark)) =_E \quad [\text{CON1}] \\
& \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; ((256\text{Kbps}; \checkmark \vee \checkmark) \wedge \checkmark) \vee \text{MPEG.2}; ((256\text{Kbps}; \checkmark \vee \checkmark) \wedge \checkmark)) =_E \quad [\text{CON5}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; (256\text{Kbps}; \checkmark \vee \checkmark) \vee \text{MPEG.2}; (256\text{Kbps}; \checkmark \vee \checkmark))
\end{aligned} \tag{3.3}$$

$$\begin{aligned}
& \text{H.264} \not\Rightarrow 256\text{Kbps in } P_2 =_E \tag{3.3} \\
& \text{H.264} \not\Rightarrow 256\text{Kbps in } (\text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; (256\text{Kbps}; \checkmark \vee \checkmark) \vee \text{MPEG.2}; (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{EXCL1}] \\
& \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; \text{H.264} \not\Rightarrow 256\text{Kbps in } ((\text{H.264}; (256\text{Kbps}; \checkmark \vee \checkmark) \vee \text{MPEG.2}; (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{EXCL4}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; ((\text{H.264} \not\Rightarrow 256\text{Kbps in } \text{H.264}; (256\text{Kbps}; \checkmark \vee \checkmark)) \vee \\
& \quad \quad (\text{H.264} \not\Rightarrow 256\text{Kbps in } \text{MPEG.2}; (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{EXCL2}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; ((\text{H.264}; ((256\text{Kbps}; \checkmark \vee \checkmark) \setminus 256\text{Kbps})) \vee \\
& \quad \quad (\text{H.264} \not\Rightarrow 256\text{Kbps in } \text{MPEG.2}; (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{EXCL1}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; ((\text{H.264}; ((256\text{Kbps}; \checkmark \vee \checkmark) \setminus 256\text{Kbps})) \vee \\
& \quad \quad (\text{MPEG.2}; \text{H.264} \not\Rightarrow 256\text{Kbps in } (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{FORB1}] \quad [\text{FORB3}] \quad [\text{FORB5}] \\
& \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; \checkmark \vee (\text{MPEG.2}; \text{H.264} \not\Rightarrow 256\text{Kbps in } (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{EXCL3}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; ((\text{H.264}; ((256\text{Kbps}; \checkmark \vee \checkmark) \setminus 256\text{Kbps})) \vee \\
& \quad \quad (\text{MPEG.2}; \text{H.264} \not\Rightarrow 256\text{Kbps in } (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{FORB1}] \quad [\text{FORB3}] \quad [\text{FORB5}] \\
& \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; \checkmark \vee (\text{MPEG.2}; \text{H.264} \not\Rightarrow 256\text{Kbps in } (256\text{Kbps}; \checkmark \vee \checkmark))) =_E \quad [\text{EXCL3}] \quad [\text{EXCL4}] \quad [\text{EXCL5}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; \checkmark \vee (\text{MPEG.2}; ((256\text{Kbps}; \checkmark) \setminus \text{H.264} \vee \checkmark))) =_E \quad [\text{FORB2}] \quad [\text{FORB3}] \quad [\text{FORB5}] \\
& \quad \text{VSS}; \text{TBR}; \text{VCC}; 720\text{Kbps}; (\text{H.264}; \checkmark \vee (\text{MPEG.2}; (256\text{Kbps}; \checkmark \vee \checkmark)))
\end{aligned} \tag{3.4}$$

Figura 3.15: Reglas de deducción aplicadas a VSS (1 de 2).

Finalmente, se presenta el *proceso de deducción* inducido por los axiomas en la figura 3.15 y la figura 3.16. En esta figura se puede observar cómo la expresión inicial P_I se transforma

hasta obtener una forma pre-normal (ver figura 3.16, ecuación 3.5):

$$\text{VSS; TBR; VCC; 720Kbps; } ((\text{H.264; } \checkmark \vee \text{MPEG.2; 720Kbps; (256Kbps; } \checkmark \vee \checkmark))$$

Este término no está en forma normal por dos razones. La primera es que no está establecido un orden entre las características. Sin embargo, se puede asumir el siguiente orden: $\text{VSS} < \text{TBR} < \text{VCC} < \text{720Kbps} < \text{H.264} < \text{MPEG.2} < \text{256Kbps}$. Además, la repetición de la característica 720Kbps en el término no está permitido en las formas normales. De esta forma, obtenemos la forma normal al aplicar las reglas en la ecuación 3.6, y el término resultante es:

$$\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; (256Kbps; } \checkmark \vee \checkmark))$$

$$\begin{aligned}
P_I &= \text{MPEG.2} \Rightarrow \text{720Kbps in H.264} \not\Rightarrow \text{256Kbps in } P_2=E & (3.4) \\
\text{MPEG.2} \Rightarrow \text{720Kbps in VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; (256Kbps; } \checkmark \vee \checkmark)) & [\text{REQ1}] \\
\text{VSS; TBR; VCC; 720Kbps; MPEG.2} \Rightarrow \text{720Kbps in (H.264; } \checkmark \vee \text{MPEG.2; (256Kbps; } \checkmark \vee \checkmark)) & [\text{REQ4}] \\
\text{VSS; TBR; VCC; 720Kbps; ((MPEG.2} \Rightarrow \text{720Kbps in H.264; } \checkmark) \vee (\text{MPEG.2} \Rightarrow \text{720Kbps in MPEG.2; (256Kbps; } \checkmark \vee \checkmark))) & [\text{REQ1}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; (MPEG.2} \Rightarrow \text{720Kbps in } \checkmark) \vee (\text{MPEG.2} \Rightarrow \text{720Kbps in (256Kbps; } \checkmark \vee \checkmark))) & [\text{REQ5}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee (\text{MPEG.2} \Rightarrow \text{720Kbps in MPEG.2;)) & [\text{REQ2}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; ((256Kbps; } \checkmark \vee \checkmark) \Rightarrow \text{720Kbps))} & [\text{MAND5}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; ((256Kbps; } \checkmark \Rightarrow \text{720Kbps) } \vee (\checkmark \Rightarrow \text{720Kbps)))} & [\text{MAND2}] \quad [\text{MAND3}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; 720Kbps; (256Kbps; } \checkmark \vee \checkmark))=E & [\text{PRE2}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; 720Kbps; (256Kbps; } \checkmark \vee \checkmark)) & \\
\end{aligned} \tag{3.5}$$

$$\begin{aligned}
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; 720Kbps; (256Kbps; } \checkmark \vee \checkmark))=E & [\text{PRE1}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{720Kbps; MPEG.2; (256Kbps; } \checkmark \vee \checkmark))=E & [\text{PRE3}] \\
\text{VSS; TBR; VCC; (720Kbps; H.264; } \checkmark \vee \text{720Kbps; 720Kbps; MPEG.2; (256Kbps; } \checkmark \vee \checkmark))=E & [\text{PRE5}] \\
\text{VSS; TBR; VCC; (720Kbps; H.264; } \checkmark \vee \text{720Kbps; MPEG.2; (256Kbps; } \checkmark \vee \checkmark))=E & [\text{PRE5}] \\
\text{VSS; TBR; VCC; 720Kbps; (H.264; } \checkmark \vee \text{MPEG.2; (256Kbps; } \checkmark \vee \checkmark)) & \\
\end{aligned} \tag{3.6}$$

Figura 3.16: Reglas de deducción aplicadas a VSS (2 de 2).

3.10. Implementación de SPLA

En esta sección se presenta la implementación de SPLA⁶. El núcleo de la herramienta está desarrollada en JAVA y su licencia es GPL v3⁷. Esta herramienta está compuesta por un conjunto de módulos que serán descritos en las secciones 3.10.1 y 3.10.2.

Para probar su aplicabilidad, se han generado una serie de SPLs utilizando el generador de modelos de características BeTTy⁸ [96]. En BeTTy, los modelos son generados tras configurar una serie de parámetros de entrada. Los valores utilizados en los parámetros de configuración se detallan a continuación:

- El porcentaje de restricciones es de un 30 %.
- La probabilidad de que exista una característica obligatoria es de 0.25.
- La probabilidad de que una característica esté dentro de una relación de *selección única* es de 0.5.

3.10.1. Módulo de factibilidad

Este módulo procesa la factibilidad de una SPL utilizando los resultados de la sección 3.8. De esta forma, se computa la fórmula asociada a una SPL. Seguidamente, utilizando un *SAT solver*, se verifica la factibilidad de la fórmula. Dado que la herramienta ha sido desarrollada en JAVA, se ha utilizado Sat4j⁹.

Los experimentos fueron ejecutados con un número distinto de características, el cual varía de 1000 to 13500. La tabla 3.17 muestra el tiempo requerido para procesar la satisfactibilidad de una SPL en función del número de características en el modelo.

3.10.2. Módulo denotacional

Este módulo procesa los productos de un término SPLA de acuerdo a las reglas de la semántica denotacional.

⁶El código fuente puede descargarse desde la dirección: <http://ccamacho.github.io/phd/spla/>.

⁷Descripción de la licencia en la siguiente URL: <http://www.gnu.org/copyleft/gpl.html>.

⁸BeTTy puede ser descargado desde: <http://www.isa.us.es/betty/betty-online>.

⁹Sitio web oficial de Sat4j: <http://www.sat4j.org/>.

Características	Tiempo (ms.)	Características	Tiempo (ms.)
01000	67	07500	529
01500	71	08000	556
02000	105	08500	682
02500	140	09000	428
03000	164	09500	639
03500	153	10000	620
04000	193	10500	396
04500	410	11000	513
05000	331	11500	575
05500	339	12000	552
06000	441	12500	413
06500	289	13000	513
07000	369	13500	503

Tabla 3.17: Benchmark de factibilidad.

Para probar la aplicabilidad de este módulo se realizó un experimento empleando un número de características, variable, de 50 a 300. Los resultados son presentados en la tabla 3.18. Esta tabla está compuesta por tres columnas de valores, la primera muestra el número de características, la segunda contiene el tiempo requerido para completar el experimento y la tercera muestra el número de productos necesarios para obtener un resultado, siendo una cota inferior del número total de características del modelo, con la propiedad de que si es 0, entonces el modelo no es factible. El símbolo guion (-) en la última columna indica que no se obtuvo ninguna respuesta tras 15 minutos de ejecución.

Características	Tiempo (ms.)	Productos	Características	Tiempo (ms.)	Productos
50	6	48	180	744	6384
60	25	108	190	1390	7232
70	15	104	200	960000	-
80	8	31	210	97770	800544
90	38	0	220	263	51
100	55	1404	230	47	8
110	13	12	240	65	29
120	2139	1	250	191	5920
130	511	24802	260	205	7296
140	7	6	270	250	4301
150	786126	1312848	280	960000	-
160	136	5670	290	65	3
170	42	398	300	960000	-

Tabla 3.18: Benchmark denotacional.

Capítulo 4

SPLA^C: Extensión de SPLA para representar costes

“Before software can be reusable it first has to be usable.”

Ralph Johnson

Contenido

4.1. Función de coste	94
4.2. Transiciones con coste	96
4.2.1. Nueva regla [req2]	99
4.2.2. Ejemplo de ejecución de reglas	102
4.3. Chef.io	106
4.4. Cálculo de coste en listas de ejecución para chef.io	109
4.4.1. Organización de la línea de productos	109
4.4.2. Especificación del coste de las características	113
4.4.3. Ejecución y resultados	116
4.5. Implementación	117
4.5.1. Consideraciones	117

En este capítulo se describe la extensión de SPLA que permite representar el coste de un producto utilizando las reglas semánticas descritas en la sección 3.3, esta extensión será conocida como SPLA^C.

La definición de la función que permite determinar el coste de agregar una característica cualquiera a un producto en construcción se presenta en la sección 4.1. Una vez definido la información que se desea generar con esta función de coste, se muestra la extensión de las reglas de la semántica operacional de SPLA en la sección 4.2. Es importante destacar que

la consistencia de esta extensión, se basa en la consistencia de las definiciones planteadas en el capítulo anterior. Una vez descrito el formalismo asociado a la extensión que define la función de coste, la sección 4.3 presenta una introducción sobre el *software* para gestionar configuraciones llamado *Chef*. Este *software*, ha sido empleado para llevar a cabo el caso de estudio, mostrado en la sección 4.4, primero utilizando FODA y luego SPLA. La sección 4.5 describe la implementación del formalismo presentado en este capítulo, así como distintas técnicas que permiten optimizar el tiempo de ejecución del mismo. Estas técnicas no están relacionadas a la implementación en sí, como puede ser el análisis previo de los términos, tal que sean eliminados aquellos operadores que no sean necesarios para modelar componentes funcionales del sistema.

4.1. Función de coste

Agregar un componente de *software* a un producto podría, o no, suponer un coste. Sin embargo, este coste no es constante y dependerá de su relación con otros componentes del producto. En este trabajo de investigación se muestran los componentes de software, como características y los productos en construcción, como trazas. La función de coste definida recibe dos parámetros, una lista, o traza, con aquellas características que ya han sido procesadas y la característica actual a ser procesada. Una vez obtenidos estos dos parámetros, se calcula el coste de agregar la característica actual a la traza de características ya procesadas.

Se asume que el coste será representado por números naturales, de igual manera, los resultados obtenidos serán equivalentes si se utilizan números reales no negativos para representar el coste. En este caso, se utilizan números enteros en vez de números reales, ya que éstos son más complejos computacionalmente para ser procesados, ya que utilizar números naturales no limita la capacidad de expresión del modelo en casos prácticos.

En algunos casos, es posible que el coste de producir una característica no pueda ser determinado a partir de su traza. Por ejemplo, antes de instalar un componente es necesario

instalar sus dependencias. Sin embargo, la función de coste no permite determinar el coste de incluir estas dependencias. Para ello, se introduce un nuevo símbolo, \perp , para poder representar esta indefinición. Se considera el conjunto $\mathbf{N}_\perp = \mathbf{N} \cup \{\perp\}$, donde serán extendidas las operaciones aritméticas al símbolo de la siguiente forma: $\forall x \in \mathbf{N}_\perp : x + \perp = \perp + x = \perp$. Además, se asume que $x \leq \perp$ para cualquier $x \in \mathbf{N}_\perp$.

Definición 4.1.1 Una *función de coste* es una función:

$$c : \mathcal{F}^* \times \mathcal{F} \mapsto \mathbf{N}_\perp$$

□

Ejemplo 4.1.2 Considerando el término $P = A; (B; \checkmark \vee C; (\bar{D}; \checkmark \wedge E; \checkmark))$, el conjunto de productos de P es:

$$\text{prod}(P) = \{\{A, B\}, \{A, C, E\}, \{A, C, D, E\}\}$$

Si se considera el orden de las características en los productos, se puede observar que A es siempre la primera característica producida, luego B o C y, finalmente si ha sido procesada C con anterioridad, D y E . Sin embargo, el formalismo no establece un orden entre D y E . Por ello, se puede considerar la siguiente función de coste:

$$\begin{array}{ll} s = \text{Traza con características procesadas.} & \\ x = \text{Característica actual a ser procesada.} & c(s, x) = \begin{cases} \perp & \text{si } x = B \text{ y } A \notin s \\ 2 & \text{si } x = D \text{ y } E \notin s \\ 1 & \text{en caso contrario} \end{cases} \end{array}$$

La función de coste es definida arbitrariamente de acuerdo al análisis de la SPL. Para el ejemplo actual han sido definidas tres reglas, donde el coste será 1, 2 o \perp dependiendo de las condiciones que se cumplan. Aunque no existe un orden preestablecido para producir las características D y E , producir D antes que E es más costoso que producir E antes que D . Es importante destacar que la función de coste define valores para trazas que pueden no ser generadas por el término P . Por ejemplo, $c(\epsilon, B)$ o $c(B, A)$. Las funciones de coste pueden ser independientes de los términos. □

4.2. Transiciones con coste

El sistema de etiquetado de transiciones ha sido definido para procesar el coste empleando grafos. De esta forma, cada nodo se corresponde a un estado dentro del sistema de etiquetado de transiciones, obtenido a partir de un término $P \in \text{SPLA}$. De manera intuitiva, cada estado se corresponde a una fase del proceso de construcción del producto. Es necesario almacenar en cada estado los componentes que han sido añadidos al producto (traza), así como el coste actual de ese producto en construcción. Cada nodo está representado por la 3-tupla (P, s, n) , donde P es el término SPLA evaluado, s es una traza de características ya procesadas y n es el número natural que indica el coste actual del producto en construcción.

Definición 4.2.1 Dado el término $P \in \text{SPLA}$ y la función de coste \mathbf{c} , el *grafo* que representa el *coste* de P , con respecto a \mathbf{c} , es el menor grafo (N, \rightarrow) que satisfaga:

[ini] $(P, \epsilon, 0) \in N$.

[cost1] Si $(Q, s, n) \in N$, $Q \xrightarrow{\mathbf{A}} Q_1$ y $\mathbf{A} \notin s$, entonces

$$(Q_1, s\mathbf{A}, n + \mathbf{c}(s, \mathbf{A})) \in N \text{ y } (Q, s, n) \rightarrow (Q_1, s\mathbf{A}, n + \mathbf{c}(s, \mathbf{A})).$$

[cost2] Si $(Q, s, n) \in N$, $Q \xrightarrow{\mathbf{A}} Q_1$ y $\mathbf{A} \in s$, entonces

$$(Q_1, s\mathbf{A}, n) \in N \text{ y } (Q, s, n) \rightarrow (Q_1, s\mathbf{A}, n).$$

□

La regla [ini] indica que inicialmente no hay características computadas y el coste es 0. La regla [cost1] hace referencia al caso donde la característica que se procesa es un término SPLA, la cual es agregada a la traza s y el coste de producir esta característica se incrementa al coste acumulado del producto. Esta condición aplica sólo si \mathbf{A} no está presente en s . La regla [cost2] indica que si la característica ya fue procesada, entonces no se modifica el coste.

La siguiente proposición indica que una vez llegada a una configuración, o estado, que no es factible en términos de coste, no tiene sentido seguir procesando características. En este caso, esta configuración se representa con \perp .

Proposición 4.2.2 Dados los términos $P, Q \in \text{SPLA}$ y la traza $s = A_1 \cdots A_l \in \mathcal{F}^*$, entonces $P \xrightarrow{s} Q$, si y solo si existen $n_1, \dots, n_l \in \mathbb{N}_\perp$ y $Q_0, Q_1 \dots Q_l$ tal que

$$P = Q_0, Q = Q_l \text{ y}$$

$$(Q_0, \epsilon, 0) \rightarrow (Q_1, A_1, n_1) \rightarrow \cdots \rightarrow (Q_l, A_1 A_2 \cdots A_l, n_l)$$

Además, si existe $1 \leq i \leq l$ tal que $n_i = \perp$, entonces $n_j = \perp$ para $i \leq j \leq l$. \square

Lema 4.2.3 Dado un término $P \in \text{SPLA}$, una función de coste c y una traza s , existen dos posibles cómputos tal que:

$$P = P_0 \xrightarrow{A_1} P_1 \cdots \xrightarrow{A_n} P_n \quad y \quad P = Q_0 \xrightarrow{A_1} Q_1 \cdots \xrightarrow{A_n} Q_n$$

con $s = A_1 \cdots A_n$. Entonces, si se consideran las siguientes transiciones de coste:

$$(P_0, \epsilon, 0) \rightarrow (P_1, A_1, c_1) \cdots \rightarrow (P_n, s, c_n) \quad y \quad (Q_0, \epsilon, 0) \rightarrow (Q_1, A_1, c'_1) \cdots \rightarrow (Q_n, s, c'_n)$$

de esta manera se obtiene que $c_n = c'_n$.

Para realizar la demostración de este lema sólo es necesario tener en cuenta que el coste depende de la traza y no de los término intermedios. \square

Definición 4.2.4 Considerando las transiciones de coste que procesen la traza s . Dado un término $P \in \text{SPLA}$, una función de coste c y un traza $s \in \text{tr}(P)$, se define el coste de producir s , escrito como $\text{tc}(P, s)$ de la siguiente manera:

$$(P_0, \epsilon, 0) \rightarrow (P_1, \mathbf{A}_1, c_1) \cdots (P_n, s, c_n) \quad P_n \xrightarrow{\checkmark} \mathbf{nil}$$

de esta manera se obtiene que $\mathbf{tc}(P, s) = c_n$.

□

El coste de producir un producto no es único, ya que éste dependerá de la traza que lo genera. Por ello, es necesario considerar el conjunto del coste para cualquier producto.

Definición 4.2.5 Dada una función de coste \mathbf{c} , se considera la función $\mathbf{c}_{\text{SPLA}} : \text{SPLA} \times \mathcal{P}(\mathcal{F}^*) \mapsto \mathcal{P}(\mathbb{N}_\perp)$ definida como:

$$\mathbf{c}_{\text{SPLA}}(P, p) = \{n \mid n = \mathbf{tc}(P, s) \text{ y } [s] = p\}$$

□

Es importante destacar que si $p \notin \mathbf{prod}(P)$, se obtiene $\mathbf{c}_{\text{SPLA}}(P, p) = \emptyset$. Por lo tanto, es posible comparar dos SPLs. Una SPL es mejor que otra, si puede producir los mismos productos con un coste inferior.

Definición 4.2.6 Dados los términos $P, Q \in \text{SPLA}$ y una función de coste \mathbf{c} , decimos que P es mejor que Q con respecto a \mathbf{c} , escrito como $P \leq_{\mathbf{c}} Q$ si y solo si $\mathbf{prod}(Q) \subseteq \mathbf{prod}(P)$ y para cualquier $p \in \mathbf{prod}(Q)$ se cumple:

$$\min\{n \mid n \in \mathbf{c}_{\text{SPLA}}(P, p)\} \leq \min\{n \mid n \in \mathbf{c}_{\text{SPLA}}(Q, p)\}$$

□

Las condiciones $p \in \mathbf{prod}(Q)$ y $\mathbf{prod}(Q) \subseteq \mathbf{prod}(P)$ implican que si los conjuntos involucrados $\{n \mid n \in \mathbf{c}_{\text{SPLA}}(P, p)\}$ y $\{n \mid n \in \mathbf{c}_{\text{SPLA}}(Q, p)\}$ no son vacíos, existe un mínimo entre ambos conjuntos.

4.2.1. Nueva regla [req2]

En este punto, se discuten los cambios realizados en la semántica operacional. En particular, el cambio a realizar sobre la regla [req2] con respecto a la versión original (ver figura 3.3). La regla [req2] original presenta algunos problemas al computar el coste.

A partir de este momento, la regla [req2] será actualizada y referenciada como la nueva regla. Esta regla, obliga a que al ejecutarse una restricción de requerimiento, ambas características sean procesadas.

$$\begin{array}{ccc} \text{[req2']} & \frac{P \xrightarrow{A} P_1}{A \Rightarrow B \text{ in } P \xrightarrow{A} P_1 \Rightarrow B} & \text{[req2]} \quad \frac{P \xrightarrow{B} P_1}{B \Rightarrow A \text{ in } P \xrightarrow{A} B; P_1} \end{array}$$

Regla de la semántica operacional original. Nueva regla para la semántica operacional.

Ejemplo 4.2.7 Dada la función de coste del ejemplo 4.1.2 y el término $P = B \Rightarrow A \text{ in } B; \checkmark$, al aplicar las reglas de transición [ini], [cost1] y [cost2] de la figura 3.3 se obtienen las siguientes transiciones:

$$(B \Rightarrow A \text{ in } B; \checkmark, \epsilon, 0) \rightarrow (\checkmark \Rightarrow A, B, \perp)$$

Entonces, el coste de producir el producto [BA] es \perp . □

De manera intuitiva, el operador de *requerimiento* establece una precondition entre las características. Precisamente, $B \Rightarrow A \text{ in } P$ indica que A es una precondition de B , esta es la justificación para modificar la regla anterior [req2'] por la nueva [req2].

Este cambio no es relevante desde el punto de vista teórico, ya que la proposición 4.2.9 muestra que no se altera la semántica original. Para justificar esta afirmación se consideran dos sistemas de transiciones, uno con la regla antigua [req2'] y otro con la regla nueva [req2]. Para diferenciar los sistemas de transición, las transiciones del antiguo modelo serán descritas como $P \xrightarrow{A} \circ Q$, mientras que en el nuevo modelo se representarán como $P \xrightarrow{A} \textcolor{blue}{n} Q$. En el siguiente lema y en la proposición 4.2.9, se considera $\text{prod}_n(P)$ y $\text{tr}_n(P)$ como el

conjunto de productos y trazas de P obtenidas mediante el nuevo sistema de transiciones, mientras $\text{prod}_o(P)$ y $\text{tr}_o(P)$ se corresponden a aquellos conjuntos generados por el sistema de transiciones antiguo, descrito en la figura 3.3.

Lema 4.2.8 Dado un producto $P \in \text{SPLA}$, las características $A, B \in \mathcal{F}$ y la traza válida $s \in \mathcal{F}^*$:

1. Si $s \in \text{tr}_o(A \Rightarrow P)$, entonces cualquier traza $s \in \text{tr}_o(P)$ y $A \in [s]$ o existe $s' \in \mathcal{F}^*$, tal que $s = s' \cdot A$ y $s' \in \text{tr}_o(P)$.
2. Si $[s] \in \text{prod}_o(P)$ y $A \in [s]$, entonces $[s] \cup \{B\} \in \text{prod}_o(A \Rightarrow B \text{ in } P)$.

Demostración: Considerando la parte 1 del lema 4.2.8, se procede por inducción del tamaño de s ($|s|$) y pueden darse 3 casos:

- $|s| = 0$: No es posible, ya que no hay reglas que permitan al operador de *obligatoriedad* producir el símbolo \checkmark . Entonces, el caso base es $|s| = 1$.
- $|s| = 1$: La única posibilidad consiste en aplicar las reglas **[mand1]** para obtener $P \Rightarrow A \xrightarrow{A}_n \checkmark$, o **[mand2]** para obtener $P \Rightarrow A \xrightarrow{A}_n P_1 \xrightarrow{\checkmark}_n \text{nil}$. En el primer caso es suficiente considerar $s' = \epsilon$ para obtener el resultado y en el segundo caso $s = A$ y $A \in \text{tr}_o(P)$.
- $|s| > 1$: Existen 3 casos, dependiendo de la regla que se aplique primero.
 - Si la primera regla es **[mand1]**, el resultado se obtiene como se describió anteriormente.
 - Si la primera regla es **[mand2]**, se tiene que $P \Rightarrow A \xrightarrow{A}_n P_1$ y $s = A \cdot s_1$, donde $s_1 \in \text{tr}_o(P_1)$. Por la premisa de **[mand2]**, se tiene que $P \xrightarrow{A}_n P_1$, entonces $s \in \text{tr}_o(P)$.

- Finalmente, si la primera regla es **[mand3]**, se obtiene $P \Rightarrow A \xrightarrow{B}_n P_1 \Rightarrow A$.
Entonces $s = B \cdot s_1$ con $s_1 \in \text{tr}_o(P_1 \Rightarrow A)$. Por la premisa en esta regla, se obtiene $P \xrightarrow{B}_n P_1$ y por la hipótesis de inducción, se obtiene el resultado.

La demostración de la parte 2 es consecuencia del resultado de la consistencia y completitud de la semántica denotacional mostrada en el capítulo anterior.

□

Proposición 4.2.9 Dado un producto $P \in \text{SPLA}$, entonces $\text{prod}_n(P) = \text{prod}_o(P)$.

Demostración: Inicialmente se considera $p \in \text{prod}_o(P)$. Por definición existe algún s , tal que $s \in \text{tr}_o(P)$ y $p = [s]$. Se demuestra que $p \in \text{prod}_n(P)$ por inducción del tamaño de s .

- $|s| = 0$: Entonces $P \xrightarrow{\check{}} \text{nil}$. Sin embargo, dado que la regla **[req2]** no permite esa transición, se obtiene que $P \xrightarrow{\check{}}_n \text{nil}$. Entonces, $s \in \text{tr}_n(P)$ y, por lo tanto, $p \in \text{prod}_n(P)$.
- $|s| > 0$: La regla aplicada para generar la primera transición en todos los casos, es la regla **[req2']**. El resultado es trivial por inducción ya que las reglas coinciden en ambos sistemas de transición. Suponiendo que la regla para obtener la primera transición de s es **[req2']**, se obtiene: $P = A \Rightarrow B \text{ in } P_1$, $P_1 \xrightarrow{A}_o Q$, $P \xrightarrow{A}_o Q \Rightarrow B$, $s = A \cdot s_1$ (donde $s_1 \in \text{tr}_o(Q \Rightarrow B)$) y $p = \{A\} \cup [s_1]$. Dado que $s_1 \in \text{tr}_o(Q \Rightarrow B)$, por el lema 4.2.8 (ver parte 1), existen dos casos:

1. $s_1 \in \text{tr}_o(Q)$ y $B \in [s_1]$;
2. existe s'_1 , tal que $s'_1 = s_1 \cdot B$ y $s'_1 \in \text{tr}_o(Q)$.

En el caso (1), por la hipótesis de inducción, se obtiene que $s_1 \in \text{tr}_n(Q)$. Entonces $B \cdot s_1 \in \text{tr}_o(B; Q)$. Desde $B \in [s_1]$ se obtiene $[B \cdot s_1] = [s_1]$, entonces $[s_1] \in \text{prod}_n(B; Q)$.

En el caso (2), por inducción $s'_1 \in \text{tr}_n(Q)$. Entonces $B \cdot s'_1 \in \text{tr}_n(B; Q)$ y $[s_1] = [s'_1] \cup \{B\}$. Así, es posible concluir también en el caso $[s_1] \in \text{prod}_n(B; Q)$. En el nuevo sistema de

transiciones, se puede aplicar **[req2]** para obtener $P \xrightarrow{A}_n B; Q$, entonces $\{A\} \cup [s_1] \in \text{prod}_n(P)$.

Considerando $p \in \text{prod}_n(P)$, entonces existe una traza $s \in \text{tr}_n(P)$, tal que $p = [s]$. Se demuestra que $p \in \text{prod}_o(P)$ por inducción del tamaño de s . El caso base es como el caso anterior. El caso inductivo se demuestra de igual manera considerando la primera regla aplicada para derivar s y, como ocurre anteriormente, todos los casos son triviales a excepción del caso cuando la primera regla aplicada es **[req2']**. En este caso particular se tiene que $P = A \Rightarrow B \text{ in } P_1$, $P_1 \xrightarrow{A}_n P'_1$ y $P \xrightarrow{B}_n A; P'_1$. Así, existe una traza s_1 tal que $s = B \cdot A \cdot s_1$ y $[A \cdot s_1] \in \text{prod}_n(P_1)$. Por la hipótesis de inducción se obtiene $[A \cdot s_1] \in \text{prod}_o(P_1)$ y, entonces, por el lema 4.2.8 (ver parte 2), $[s] = [A \cdot s_1] \cup \{B\} \in \text{prod}_o(A \Rightarrow B \text{ in } P_1)$.

□

Con la regla **[req2]**, es posible procesar el coste de los productos del término del ejemplo 4.2.7. Es importante destacar que un producto puede ser obtenido de distintas trazas: $P = \{A, B\} = [AB] = [BA]$.

4.2.2. Ejemplo de ejecución de reglas

En esta sección se presenta un ejemplo para ilustrar cómo se procesa el coste en el modelo. Para poder mostrar su aplicación, se define una SPL que está formada por cinco componentes funcionales (A, B, C, D y E). Se considera el término SPLA del ejemplo 4.1.2:

$$P = A; (B; \checkmark \vee C; (\bar{D}; \checkmark \wedge E; \checkmark))$$

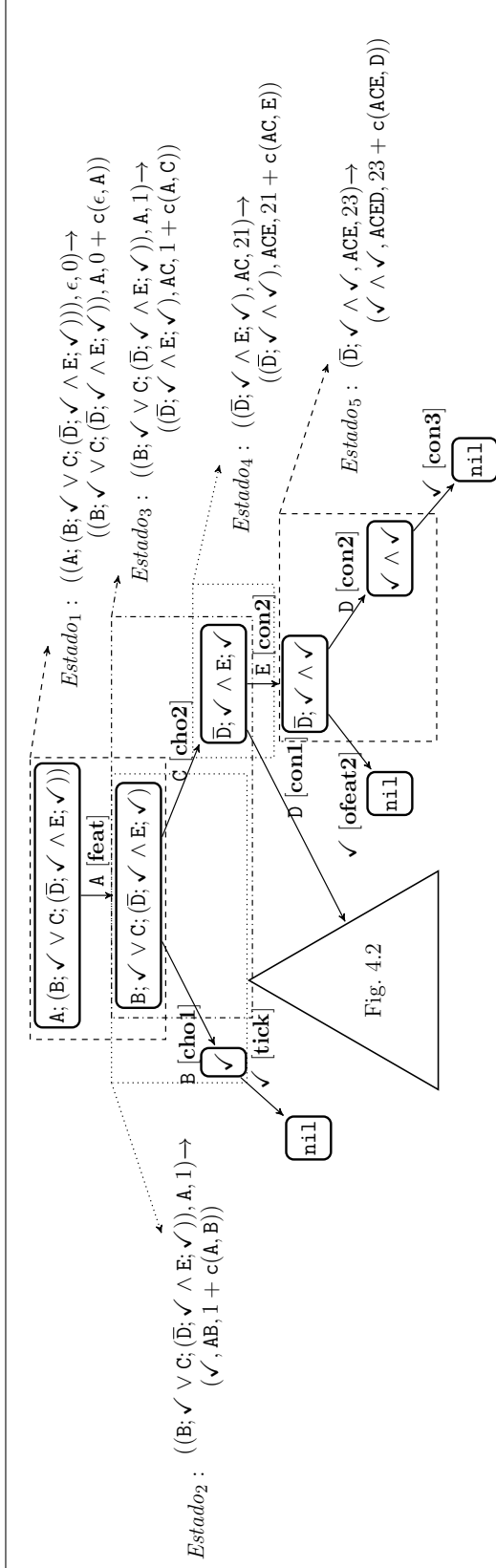


Figura 4.1: Ejemplo del modelado del coste en FODA (1 de 2).

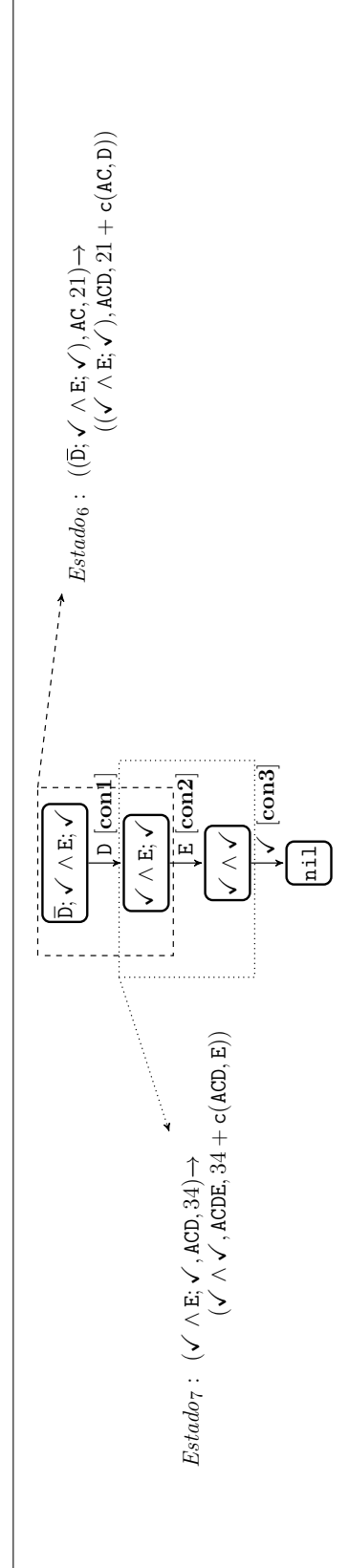


Figura 4.2: Ejemplo del modelado del coste en FODA (2 de 2).

c	A	B	C	D	E
ϵ	1	1	1	1	1
A	\perp	22	20	125	134
AC	12	15	\perp	13	2
ACE	\perp	11	\perp	332	\perp
ACD	\perp	132	\perp	\perp	581

Tabla 4.3: Función de coste

Se ha analizado este término empleando la función de coste general con los valores de la tabla 4.3. Esta tabla muestra el coste de algunos posibles valores en $\mathcal{F}^* \times \mathcal{F}$. Se ha definido $c(s, x) = \perp$ para el resto de los pares $(s, x) \in \mathcal{F}^* \times \mathcal{F}$ que no aparecen en la tabla. Es importante destacar que la función de coste proporciona algunos valores para opciones no permitidas por el término P . Por ejemplo $c(\text{AC}, \text{B}) = 15$, indica el posible valor de producir la característica B si previamente se ha producido A y C. Esta función de coste pudo haber sido usada en términos como $Q = \text{A}; (\text{B} \wedge \text{C})$, pero en el caso del término P este valor concreto nunca será producido, ya que en P las características C y B nunca coinciden en el mismo producto.

La figura 4.1 y la figura 4.2 describen cómo se procesa P utilizando las reglas de la semántica operacional presentadas en la figura 3.3, donde el coste de procesar las reglas es aplicado, siempre y cuando, las reglas semánticas se hayan procesado.

En este ejemplo, la primera característica procesada es A, para lo cual se utiliza la regla [feat]. Una vez que esta regla se ha procesado, se aplica la regla [cost1]. Entonces, se obtiene la transición Estado_1 :

$$(\text{A}; (\text{B}; \checkmark \vee \text{C}; (\overline{\text{D}}; \checkmark \wedge \text{E}; \checkmark)), \epsilon, 0) \rightarrow (\text{B}; \checkmark \vee \text{C}; (\overline{\text{D}}; \checkmark \wedge \text{E}; \checkmark), \text{A}, 1)$$

Luego, se aplica la regla [cho1] para obtener Estado_2 :

$$(B; \checkmark \vee C; (\bar{D}; \checkmark \wedge E; \checkmark), A, 1) \rightarrow (\checkmark, AB, 23))$$

y seguidamente se aplica la regla **[cho2]** para obtener *Estado*₃:

$$(B; \checkmark \vee C; (\bar{D}; \checkmark \wedge E; \checkmark), A, 1) \rightarrow (\bar{D}; \checkmark \wedge E; \checkmark, AC, 21)$$

Como resultado, se puede confirmar que el coste de la traza **AB** es 23. Puesto que es la única manera de obtener el producto $\{A, B\}$, su coste es 23.

El término $\bar{D}; \checkmark \wedge E; \checkmark$ tiene dos posibles transiciones, una obtenida al aplicar la regla **[con1]**, lo cual lleva al término $\checkmark \wedge E$ y otra aplicando la regla **[con2]**, generando el término \bar{D} . El término $\checkmark \wedge \checkmark$ obtiene la transición de coste *Estado*₇ (ver figura 4.2):

$$(\bar{D}; \checkmark \wedge E; \checkmark, AC, 21) \rightarrow (\checkmark \wedge E; \checkmark, ACD, 34)$$

y *Estado*₄:

$$(\bar{D}; \checkmark \wedge E; \checkmark, AC, 21) \rightarrow (\bar{D}; \checkmark \wedge \checkmark, ACE, 23)$$

De este último término, es posible aplicar dos reglas: **[ofeat2]** y **[con2]**. La primera genera \checkmark , donde se obtiene el producto $\{A, C, E\}$ con coste 23. Esta es la única manera de obtener este producto. De la segunda regla se obtiene la transición de coste *Estado*₅:

$$(\bar{D}; \checkmark \wedge \checkmark, ACE, 23) \rightarrow (\checkmark \wedge \checkmark, ACED, 355)$$

Como resultado, se obtiene el producto $\{A, C, D, E\}$ con coste 355. En este caso no es la única manera de generar este producto, ya que si se selecciona la transición de coste *Estado*₆, al aplicar la regla **[con2]**, se obtiene la transición de coste:

$$(\checkmark \wedge E; \checkmark, ACD, 34) \rightarrow (\checkmark \wedge \checkmark, ACDE, 615)$$

En este caso, se obtiene el producto $\{A, C, D, E\}$ con coste 615, por lo tanto, la función c_{SPLA} se define como:

p	$c_{\text{SPLA}}(P, p)$
$\{A, B\}$	$\{23\}$
$\{A, C, E\}$	$\{23\}$
$\{A, C, E, D\}$	$\{355, 615\}$

Considerando los términos SPLA R y Q definidos como:

$$Q = A; (B; \checkmark \vee C; (\bar{D}; E; \checkmark \vee E; \checkmark)) \quad R = A; (B; \checkmark \vee C; E; \bar{D}; \checkmark)$$

Es fácil comprobar que $\text{prod}(P) = \text{prod}(Q) = \text{prod}(R)$. Sin embargo, en P , el subtérmino $\bar{D}; \checkmark \wedge E; \checkmark$ indica que no existe un orden preestablecido para producir las características E y D . Por el contrario, este orden es establecido en Q y R . Puesto que producir E antes que D tiene un coste menor que producir D antes que E , se obtiene:

$$R \leq_c P \leq_c Q$$

Esto indica que la SPL representada por R es preferible a P y Q . Además, P es preferible a Q .

4.3. Chef.io

Chef es una herramienta para la gestión de la configuración que permite orquestar el despliegue de una configuración previamente definida (*recipes* o recetas), en aquellos nodos que estén registrados en el servidor de la herramienta en cuestión [2].

Este sistema facilita el proceso de centralizar la configuración de un conjunto de nodos en un único repositorio. De esta forma, el sistema permite definir a través de código fuente, la especificación de la infraestructura.

La configuración básica de un entorno gestionado por *Chef* deberá estar formada al menos por una estación de trabajo (*workstation*), un servidor de *Chef* y un conjunto de

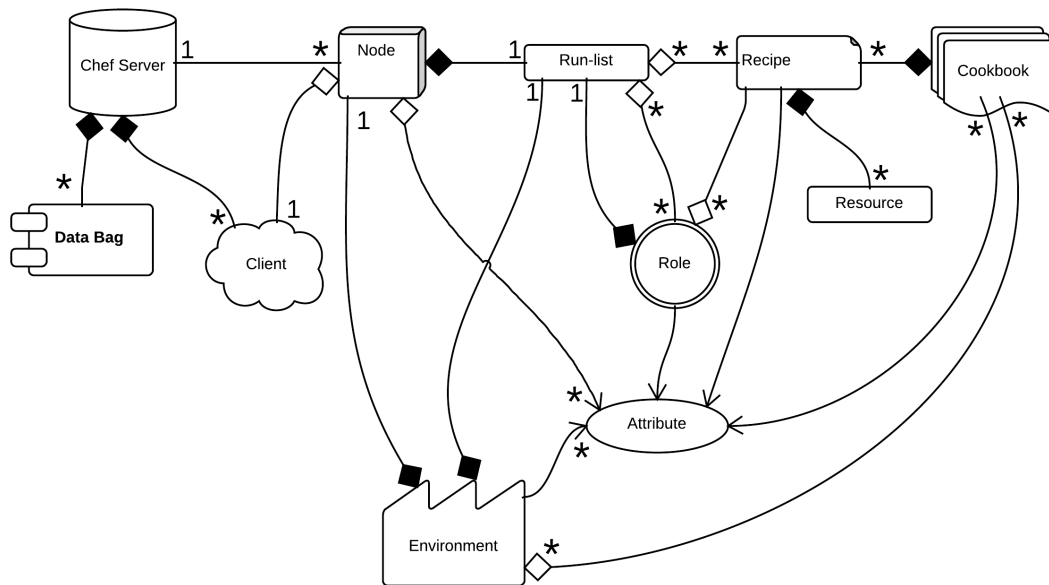


Figura 4.4: Arquitectura general del servicio de *Chef*.

nodos registrados (*nodes*). Una vez que estos nodos han ejecutado el cliente de *Chef* (*chef-client*), el servidor especificará cuáles son las recetas que deben ser ejecutadas. Las recetas son agrupadas en libros de cocina (*cookbooks*) y roles, dependiendo de la configuración del sistema. Una vez que el servidor de *Chef* calcula la secuencia de pasos para configurar el nodo, éste ejecuta los pasos correspondientes para estar en un estado deseable, en el cual, todas las recetas debieron haberse ejecutado satisfactoriamente.

En la sección 4.4 se presenta un ejemplo que ilustra cómo crear listas de ejecución (*run-list*) para el sistema de gestión de la configuración implementado con *Chef*. Inicialmente, con la finalidad de proveer información de contexto sobre el funcionamiento de *Chef*, se describen los distintos componentes del sistema de gestión de la configuración que interactúan para generar *run-lists* válidas. La figura 4.4 describe todos los componentes del servicio de *Chef*, los cuales se describen en detalle en la documentación oficial [2]:

- *Chef server*. El servidor de *Chef* funciona como un repositorio centralizado de confi-

guración. Además, almacena toda la información requerida para configurar los nodos registrados.

- *Node*. Es un recurso de cómputo registrado en el servidor de Chef.
- *Ohai*. Es una herramienta que permite obtener información del hardware del nodo registrado.
- *Run-list*. Es una lista secuencial de recetas que al ser ejecutada, configura un nodo.
- *Recipe*. Es el elemento fundamental de configuración y especifica el recurso al que se debe llamar para configurar un nodo.
- *Cookbook*. Representa un conjunto de recetas, recursos, proveedores y atributos.
- *Data-bag*. Es una variable global accesible desde el servidor de Chef.
- *Chef-client*. Es la herramienta cliente que será ejecutada en cada nodo registrado. Se encarga de calcular y ejecutar las recetas correspondientes (*run-list*).
- *Environment*. Es el vínculo lógico entre los nodos, recetas, *data-bags* y atributos. Pueden crearse para representar la organización de los entornos pertenecientes al sistema. Por ejemplo, entorno de integración continua, pruebas y producción.
- *Attribute*. Es una variable accesible dentro de la receta que podrá sobrescribir la configuración local del nodo.
- *Role*. Es una agrupación de recetas de acuerdo a su función.
- *Resource*. Define el paso requerido para ejecutar una acción. Por ejemplo, al especificar un recurso, el usuario especifica qué debe realizar la receta, pero no cómo hacerlo.

4.4. Cálculo de coste en listas de ejecución para chef.io

En esta sección se presenta un ejemplo práctico que ilustra el cálculo de la lista de tareas a ejecutar (run-list) al configurar un conjunto de servidores, es decir, establecer el orden en el que los servicios (web, base de datos y monitorización) serán instalados. La lista establecerá el orden de ejecución de las tareas, basándose en una función de coste provista por el formalismo descrito en esta sección, cuyo resultado final será generar una lista de pasos de configuración de coste mínimo.

4.4.1. Organización de la línea de productos

El modelo de características para este ejemplo se genera al analizar los componentes del sistema de gestión de la configuración, descritos en la figura 4.4, así como las características del sistema a modelar. Este sistema está compuesto por dos entornos (producción y pruebas), dos servidores para cada entorno y un conjunto de aplicaciones (servidor web, servidor de base de datos y sistema de monitorización).

Es importante determinar todas las características, así como también las relaciones entre ellas para poder especificar correctamente el sistema. El objetivo principal de este modelo es crear listas de ejecución válidas para un conjunto de nodos registrados en un servidor de Chef. A continuación se describirán los requerimientos y componentes del sistema.

- Existen dos entornos, **production** y **test**. Para poder modelar esta condición es necesario agregar restricciones al modelo de variabilidad. Estas restricciones se obtienen de la configuración del sistema. La sintaxis para describir un entorno en Chef es la siguiente:

```
name "environment_name"
description "environment_description"
cookbook OR cookbook_versions "cookbook" OR "cookbook" => "cookbook_version"
default_attributes "node" => { "attribute" => [ "value", "value", "etc." ] }
override_attributes "node" => { "attribute" => [ "value", "value", "etc." ] }
```

En este caso, es posible configurar el entorno sobrescribiendo los valores por defecto.

El entorno **production** se presenta como:

```

name "production"
description "Production environment"
cookbook_versions({
  "nginx" => "<= 1.1.0",
  "mysql" => "<= 4.2.0",
  "apache" => "= 0.4.1"
})
override_attributes ({
  "apache" => {
    "listen" => [ "80", "443" ]
  },
  "mysql" => {
    "root_pass" => "XXXXX"
  }
})

```

El entorno **test** se presenta como:

```

name "test"
description "Test environment"
cookbook_versions({
  "nginx" => "<= 2.1.0",
  "mysql" => "<= 6.2.0",
  "apache" => "= 1.4.1"
})
override_attributes ({
  "apache" => {
    "listen" => [ "8080", "4430" ]
  },
  "mysql" => {
    "root_pass" => "XXXXX"
  }
})

```

- Existen dos nodos para cada entorno, uno para albergar el servidor web y otro que contiene el servidor de base de datos. Para poder modelar esta condición, es necesario ejecutar los siguientes comandos al realizar el despliegue del cliente de Chef en los nodos:

```

knife bootstrap web.example.com -r 'role[webserver]' -e production
knife bootstrap db.example.com -r 'role[database]' -e production

```

Estos comandos configuran las recetas del rol **web** en el servidor `webserver.example.com` utilizando los atributos del entorno **production**. De manera similar, se configura el servidor `dbserver.example.com` como un servidor de base de datos en el entorno de producción.

- Seguidamente, estos nodos deben ser monitorizados. Para ello, se actualiza el comando que instala el cliente de Chef de la siguiente forma:

```

knife bootstrap webserver.example.com -r 'role[web]','role[monitor]' -e production
knife bootstrap dbserver.example.com -r 'role[db]','role[monitor]' -e production

```

En este caso se agrega el rol *monitor* para ambos nodos.

- El servidor web y el servidor de base de datos deben ser desplegados físicamente en nodos distintos (condición requerida por el usuario). Esto se consigue manteniendo los roles separados en cada nodo al instalar el cliente de Chef.
- En el entorno de producción, el servidor de base de datos debe ser MySQL y el sistema de monitorización debe ser Nagios. Para conseguir este resultado, es posible acceder a variables globales de entorno dentro de las recetas. Una vez configuradas estas variables, se pueden validar y ejecutar las acciones que correspondan.

```
if node.chef_environment == "production"
  # do something, i.e. Install MySQL, Nagios.
else
  # do other thing.
end
```

También es necesario modelar cómo los elementos comunes y variables del modelo interactúan entre sí. Estos elementos pueden ser funcionales o no-funcionales. Dada la siguiente arquitectura de *Chef*, mostrada en la figura 4.4, se obtienen las relaciones necesarias para construir el modelo de variabilidad, las cuales se muestran a continuación.

Una instalación de Chef puede tener 0 o más Roles, 0 o más *cookbooks* y al menos 1 entorno. Un nodo pertenece a sólo un entorno. Un nodo tiene 0 o más Roles. Un nodo puede tener 0 o más *cookbooks*. Un rol puede tener 0 o más *cookbooks*. Un nodo tiene una única lista de ejecución.

De esta forma, para crear el modelo de variabilidad se emplean estas relaciones y condiciones. Esta información puede traducirse directamente a un diagrama FODA (ver figura 4.5).

El diagrama FODA muestra las *run-lists* *R1*, *R2*, *R3*, *R4*, *R5* y *R6*. Por ejemplo, el contenido de la *run-list* *R4* una vez seleccionado *SQLServer*, contendrá todos los pasos requeridos para configurar un servidor de base de datos con *SQLServer*. Estas listas de ejecución (*run-lists*) son recetas que deben ser ejecutadas y no tienen influencia sobre el modelo de variabilidad. Por lo tanto, estas referencias deben ser eliminadas del modelo. El orden de ejecución de listas será tomado en cuenta al analizar el modelo de coste.

A continuación se muestra un ejemplo sobre como son especificados los *cookbooks*.

```

#
# Cookbook Name:: installers_custom
# Recipe:: sqlserver
#
#Steps
#1- Download ISO according OS (w2008 or w2012) if not downloaded.
#2- Unzip ISO if not unzipped.
#3- Install SQL Server if not installed.
#
#Step 1
case node['platform']
when 'windows'
  require 'chef/win32/version'
  windows_version = Chef::ReservedNames::Win32::Version.new
  if windows_version.windows_server_2008_r2?
    unless File.exists?('C:/en_sql_server_2008.iso')
      remote_file 'C:/en_sql_server_2008.iso' do
        source 'http://software.example.com/en_sql_server_2008.iso'
        owner 'Administrator'
      end
    end
  end

  #Step 2
  unless File.exists?('C:/SQL_SERVER')
    batch 'unzip SQL Server' do
      code <<-EOH
      7zG x -y "C:/en_sql_server_2008.iso" -o"C:/SQL_SERVER"
      .
      .
      .
    end

    #Step 3
    if File.exists?('C:/SQL_SERVER')
      batch 'unzip SQL Server' do
        code <<-EOH
        "C:/SQL_SERVER/setup.exe" /QUIET
        EOH
      end
    end
  end
end

```

La especificación del término **SPLA** ha sido extraída del diagrama descrito en la figura 4.5. Además, esta especificación puede ser traducida para crear directamente el modelo de variabilidad de la figura 4.6.

Las características **ChefServer**, **Role**, **Node** y **Environments** forman parte del análisis del dominio de la **SPL**, pero no afectan necesariamente al comportamiento funcional de los productos, ya que representan categorías de características. Como consecuencia, éstas pueden ser eliminadas sin modificar el modelo de variabilidad. El término simplificado se obtiene de procesar el término de la figura 4.6 y así obtener como resultado, el término presentado en la figura 4.7. Esta simplificación tiene un gran impacto en el desempeño de la herramienta, como se muestra en la sección 4.5.

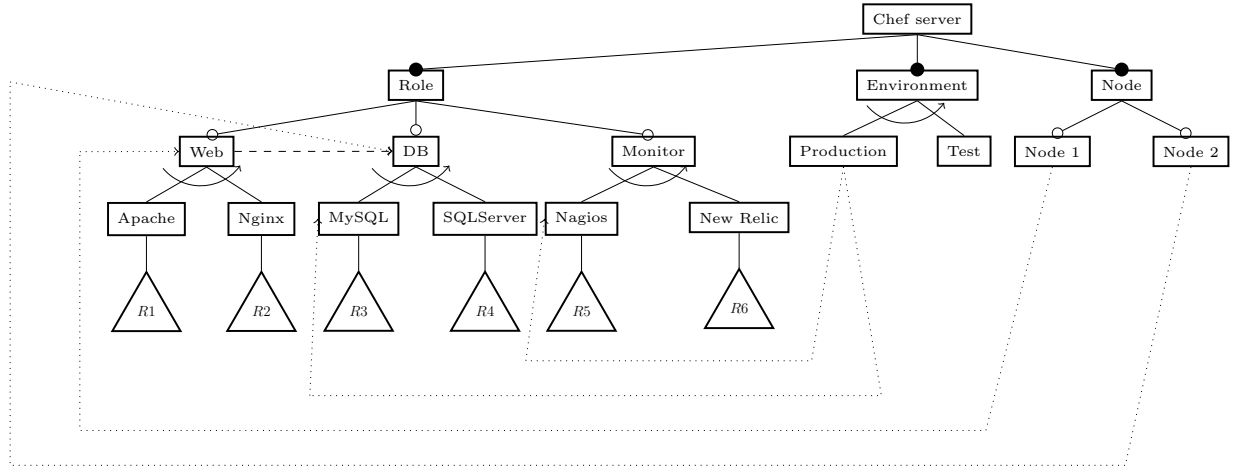


Figura 4.5: Modelo de características de chef.io.

4.4.2. Especificación del coste de las características

Una vez definido el modelo de variabilidad, es posible construir el modelo de coste. Para simplificar el modelo, se utilizará el caso del servidor de base de datos dentro del entorno de producción.

El modelo de coste se define teniendo en cuenta la complejidad de las recetas asociadas a cada nodo. El modelo de variabilidad no restringe el orden en el que las características **Nagios** y **MySQL** se agregan al producto. Sin embargo, este orden es importante. Por ejemplo, si la característica **Nagios** se agrega al producto antes que la característica **MySQL**, se ejecutará el siguiente proceso: La característica **MySQL** debe ser monitorizada. Una vez que **MySQL** es agregada, las recetas asociadas a **Nagios** deben ser recalculadas. Por el contrario, estas recetas no deben ser recalculadas si se agrega la característica **MySQL** antes que **Nagios**. De manera similar, ocurre lo mismo con las características **Nagios** y **Apache**. Finalmente obtenemos como conclusión que el servidor web y el servidor de base de datos deben estar en distintos nodos, indicando que el coste de agregar ambas características **Apache** y **MySQL**, o viceversa, al mismo producto, es \perp .

<pre> run-list = N1 ⇒ W in (N2 ⇒ D in (P ⇒ MS in (P ⇒ NA in (W ⇒ D in (S; (R; (W̄; (AP; ✓ ∨ NG; ✓) ^ D̄; (MS; ✓ ∨ SS; ✓) ^ M̄; (NA; ✓ ∨ NR; ✓)) ^ E; (P; ✓ ∨ T; ✓) ^ N; (N1; ✓ ∨ N2; ✓)))))) </pre>	<pre> S : Chef server R : Role E : Environment N : Node P : Production T : Test W : Web server D : Database server M : Monitoring AP : Apache NG : Nginx MS : MySQL SS : SQLServer NA : Nagios NE : New Relic N1 : Node 1 N2 : Node 2 </pre>
--	--

Figura 4.6: Traducción de diagrama FODA a SPLA.

$$c(s, x) = \begin{cases} Regla1 : 7 & \text{si } x = \text{MySQL y Nagios} \in s \\ Regla2 : 7 & \text{si } x = \text{Apache y Nagios} \in s \\ Regla3 : 4 & \text{si } x = \text{Nagios y MySQL} \in s \\ Regla4 : 4 & \text{si } x = \text{Nagios y Apache} \in s \\ Regla5 : \perp & \text{si } x = \text{Apache y Mysql} \in s \\ Regla6 : \perp & \text{si } x = \text{Mysql y Apache} \in s \\ Regla7 : 0 & \text{en cualquier otro caso} \end{cases}$$

- [Regla1:] Se procesa la característica **MySQL** cuando **Nagios** ya ha sido procesada. Esta regla determina que deben ejecutarse 7 recetas para tener el servicio de MySQL monitorizado con Nagios, ya que el servicio de monitorización ha sido previamente desplegado.
- [Regla2:] Se procesa la característica **Apache** cuando **Nagios** ya ha sido procesada. En este caso, se obtiene el mismo coste que Regla1, ya que deben ejecutarse 7 recetas

$run-list_{op} =$	
$N1 \Rightarrow W \text{ in } ($	$P : \text{Production}$
$N2 \Rightarrow DB \text{ in } ($	$T : \text{Test}$
$P \Rightarrow MS \text{ in } ($	$W : \text{Web server}$
$P \Rightarrow NA \text{ in } ($	$D : \text{Database server}$
$W \not\Rightarrow D \text{ in } ($	$M : \text{Monitoring service}$
$\bar{W}; (AP; \checkmark \vee NG; \checkmark)$	$AP : \text{Apache}$
\wedge	$NG : \text{Nginx}$
$\bar{D}; (MS; \checkmark \vee SS; \checkmark)$	$MS : \text{MySQL}$
\wedge	$SS : \text{SQLServer}$
$\bar{M}; (NA; \checkmark \vee NR; \checkmark)$	$NA : \text{Nagios}$
\wedge	$NE : \text{New Relic}$
$P; \checkmark \vee T; \checkmark$	$N1 : \text{Node 1}$
\wedge	$N2 : \text{Node 2}$
$\bar{N1}; \checkmark \vee \bar{N2}; \checkmark$	
)	
)	
)	
)	
)	

Figura 4.7: Término optimizado SPLA.

para tener el servidor web monitorizado.

- [Regla3:] Se procesa la característica **Nagios** y la característica **MySQL** ya ha sido procesada. En este caso la ejecución es menos costosa, ya que sólo deben ejecutarse 4 recetas para tener el servicio de MySQL monitorizado.
- [Regla4:] Se procesa cuando la característica **Nagios** es producida y **Apache** ya ha sido producida. Se obtiene el mismo coste que Regla3. Sólo deben ejecutarse 4 recetas para tener el servidor web monitoreado.
- [Regla5 y Regla6:] El servidor web y el servidor de base de datos deben desplegarse en nodos separados.
- [Regla7:] El coste de producir cualquier otra característica en cualquier otro orden es 0.

4.4.3. Ejecución y resultados

A continuación, se describe el procesamiento del término SPLA^C *run-list* para el ejemplo de esta sección. Inicialmente se genera un árbol de trazas con todos los estados válidos e inválidos. El término *run-list* genera una gran cantidad de trazas, las cuales no pueden ser representadas gráficamente. Por ello, se presentarán algunas trazas representativas después de procesar $\text{tr}(\text{run-list})$ para el nodo del servidor de base de datos en el entorno de producción

$$\text{tr}(\text{run-list}) = \left\{ \begin{array}{l} \dots \\ ['Node2', 'Prod', 'DB', 'MySQL', 'Monitor', 'NewRelic', 'Nagios'] \\ ['Node2', 'Prod', 'Monitor', 'DB', 'Nagios', 'MySQL'] \\ \dots \\ ['Node2', 'Prod', 'Monitor', 'DB', 'NewRelic', 'MySQL', 'Nagios'] \\ \dots \end{array} \right\}$$

Una vez calculado el coste de las trazas, se obtienen los correspondientes productos y su coste asociado.

p	$\text{CSPLA}(P, p)$
\dots	\dots
$\{\text{Node2}, \text{Prod}, \text{DB}, \text{MySQL}, \text{Monitor}, \text{NewRelic}, \text{Nagios}\}$	$\{4, 7\}$
$\{\text{Node2}, \text{Prod}, \text{Monitor}, \text{DB}, \text{Nagios}, \text{MySQL}\}$	$\{4, 7\}$
\dots	\dots

Es importante destacar que la lista de ejecución(*run-list*) debe ser idempotente. Esto es, que sin importar cuantas veces sea ejecutado el cliente de Chef, la salida del sistema será siempre la misma. Por ejemplo, si una receta instala, por primera vez, el servicio de MySQL, las siguientes ejecuciones de la misma serán omitidas, ya que el nodo se encuentra en el estado deseado. En este caso, el *framework* formal representa esta condición de manera intuitiva, puesto que el problema se trata como una máquina de estados determinista.

4.5. Implementación

Esta sección muestra la implementación del *framework* formal presentado en este capítulo. El *software* está licenciado bajo GPL v3¹. La herramienta está desarrollada en Python y genera una máquina de estados finitos que modela los estados válidos del sistema, es decir, genera los productos válidos. Mientras las características son procesadas, el coste y las trazas son almacenadas en cada término intermedio para su posterior uso. La implementación² contiene la documentación del API, instrucciones para su uso, ficheros de ejemplo y el código fuente.

4.5.1. Consideraciones

La implementación padece de un problema computacional clásico, el cual consiste en calcular todas las posibles trazas del modelo de variabilidad para poder así determinar qué producto es mejor que otro en función de su coste. Uno de los principales inconvenientes se encuentra en el comportamiento del operador de *paralelo* (\wedge), cuyas reglas son [con1], [con2] y [con3] de la semántica operacional.

Las trazas generadas de $P \wedge Q$ deben tener las características de P y Q en todas las posibles combinaciones. Esto crea una explosión combinatoria, ya que la combinación de k características sin repetición tiene $k!$ combinaciones distintas.

Para reducir el efecto de esta explosión combinatoria, existen algunas consideraciones que pueden ser tenidas en cuenta cuando la línea de productos está siendo diseñada, pudiendo así, optimizar su tiempo de ejecución.

Una forma de lidiar con este inconveniente consiste en realizar podas en las trazas mientras los términos son procesados. Esto se consigue por varios métodos, como por ejemplo, con el uso de *umbrales de coste*. En este caso, no serán producidos aquellos productos fuera del umbral HC (*coste mayor*) o LC (*coste menor*). También es posible limitar el cómputo

¹Más detalles en <http://www.gnu.org/copyleft/gpl.html>.

²El código fuente de la implementación puede ser descargado directamente desde la siguiente URL: <http://ccamacho.github.io/phd/splc/>.

agregando símbolos (\perp) en la función de coste, de forma que las trazas que lo contengan, nunca terminarán en productos válidos. Es importante destacar que los términos son independientes y la información referente al coste está almacenada, además, dentro de cada término. Así, el modelo puede ser ejecutado en un sistema de cómputo distribuido, compartiendo la carga de trabajo entre varios nodos.

Optimización del modelo de características

Los diagramas de FODA tienen una representación estricta, en la cual todos los elementos funcionales y no funcionales del modelo deben ser representados. Cuando se procesa el operador de *paralelo* en un término SPLA, existe una penalización de desempeño, ya que se generan n nuevos términos (uno por cada operando que pueda procesarse). Esta es una condición inherente al uso del operador de paralelo en la semántica operacional de SPLA.

Para mitigar esta condición es posible, por ejemplo, eliminar aquellos componentes no funcionales dentro del modelo, ya que éstos no tienen influencia en el comportamiento de los productos válidos que han sido producidos. Este tipo de condiciones permite optimizar el número de características en el modelo de variabilidad. Por ejemplo, en la sección 4.4 se presenta un ejemplo sobre cómo optimizar modelos que provengan de diagramas FODA, donde se muestra el término *run-list* de la figura 4.6, que resulta del análisis del diagrama de la figura 4.5. Como se ha mostrado, los productos de este término contienen las características **ChefServer**, **Roles**, **Cookbooks** y **Environments**, que son parte del análisis del dominio de la SPL. Sin embargo, no afectan necesariamente al comportamiento de los productos, ya que representan categorías de características. De esta forma, se presenta el término SPLA *run-list_{op}* en la figura 4.7, donde estas características han sido eliminadas. El conjunto de productos de ambos términos son esencialmente los mismos, ya que sólo se han eliminado características que representan categorías.

Esta simple modificación tiene un impacto importante sobre la ejecución del algoritmo. El cómputo del término *run-list* genera 97648 productos válidos en 2340.49 segundos, mientras que la ejecución de *run-list_{op}* genera 10572 productos válidos en 491.41 segundos. Este

resultado muestra que un correcto análisis de la organización de la **SPL** puede afectar directamente en la ejecución del algoritmo. En este caso particular, al eliminar los componentes no funcionales, el tiempo de ejecución mejoró un 79 % y el número de productos producidos se redujo un 89,1 %.

Cómputo distribuido

Los términos **SPLA** pueden ser procesados en sistemas distribuidos de manera natural. En este caso, todos los estados del sistema pueden ser tratados como componentes independientes, ya que su procesamiento no depende de otros (toda la información necesaria para el procesamiento de un término, está dentro del mismo). Por ejemplo, consideramos el siguiente término:

$$P = (F1; P1) \vee (F2; P2) \vee (F3; ((F4; P3) \vee (F5; P4)))$$

donde $P1$, $P2$, $P3$ y $P4$ son términos válidos **SPLA**. Estos términos pueden representar un estado terminal en el que una traza válida será producida. Procesar estos términos en un sistema distribuido puede realizarse de manera natural, ya que no hay dependencias entre estados distintos y se tendrá autocontenida toda la información necesaria para distribuir el cómputo en un sistema distribuido.

Para de ejecutar esta implementación en un sistema distribuido se utilizará **SCOOP** (del inglés, Scalable COncurrent Operations in Python) [54] el cual es un *framework*, escrito en el lenguaje de programación Python, utilizado para distribuir de manera automática tareas dinámicas. En este caso, se distribuyen de manera recursiva un conjunto de estados entre un conjunto de hilos de ejecución o *SCOOP workers*, los cuáles serán automáticamente distribuidos entre los recursos disponibles del sistema. Un *worker* es un instancia de cómputo usada para ejecutar una tarea. Múltiples *workers* se distribuyen entre los recursos del sistema distribuido para completar la tarea de forma más eficiente. En un nodo de cómputo pueden ejecutarse uno o más *workers*.

	1 Hilo	2 Hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
1 Nodo	2010.44763303	1060.80047798	586.014445066	543.712262154	521.616870165	521.292215109
2 Nodos	2059.06947899	1046.87119007	575.115453959	296.285589933	366.384442091	341.007520199
4 Nodos	2031.25184584	1093.52087283	586.344302893	320.010899067	300.745616913	382.748430014
8 Nodos	2207.35427213	1143.951792	576.370896101	495.507214785	308.374300957	287.340030909

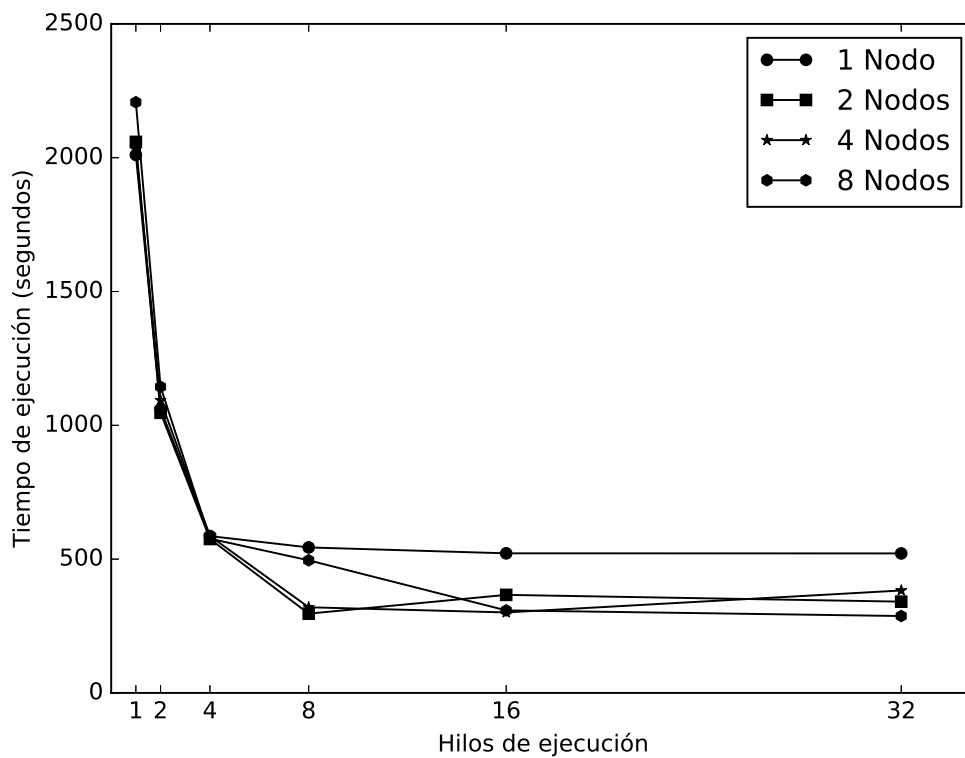


Figura 4.8: Tiempos de ejecución para 1, 2, 4 y 8 nodos y 1, 2, 4, 8, 16 y 32 hilos de ejecución.

La implementación se ha probado en un *cluster* de 8 nodos, donde cada nodo tiene la siguiente configuración:

- Sistema Operativo: Ubuntu 14.04-trusty
- RAM: 4GB
- Procesador: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
- Núcleos de CPU: 4

El término SPLA empleado para ser procesado en el sistema distribuido es *run-list* (ver figura 4.6). Este término genera 97648 productos válidos. La figura 4.8 muestra los tiempos de ejecución para procesar todos los productos de *run-list* en función de los nodos de cómputo y los hilos de ejecución. El mejor resultado se obtiene al utilizar 8 nodos y 32 hilos de ejecución (287.34 segundos). Sin embargo, la ganancia no es significativamente relevante con respecto a configuraciones que utilizan menos recursos. Por ejemplo, la ganancia de la configuración utilizando 8 nodos y 32 hilos de ejecución con respecto a la configuración que emplea 4 nodos y 16 hilos de ejecución es de sólo 13.41 segundos, esto es, una mejora de rendimiento del 4.67 % utilizando el doble de recursos de cómputo.

Umbrales

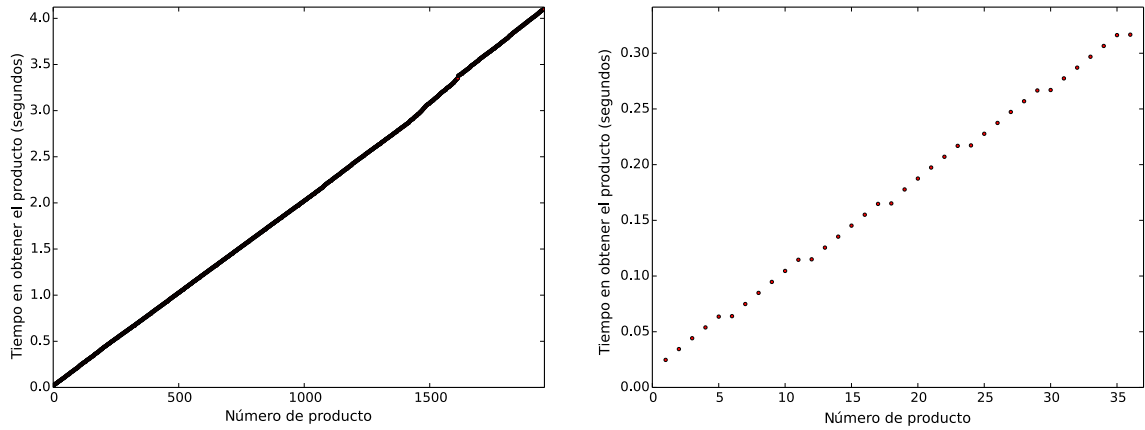


Figura 4.9: Tiempo acumulativo para el procesamiento de productos sin umbrales (izquierda) y con umbrales Min=1, Max=2 (derecha).

Es posible definir umbrales como solución para minimizar el tiempo de cómputo para procesar un término SPLA. En este caso, es posible descartar las trazas que superen un coste determinado. Por ejemplo, considerando el término

$$P = \overline{F1}; \checkmark \wedge \overline{F2}; \checkmark \wedge \overline{F3}; \checkmark \wedge \overline{F4}; \checkmark \wedge \overline{F5}; \checkmark \wedge \overline{F6}; \checkmark$$

donde el coste de producción de cada característica es siempre 1, el coste total de un producto será igual al número de características que contenga.

En este ejemplo P genera 1957 productos válidos en 4.02 segundos, estos resultados son presentados en la figura 4.9 (izquierda), sin utilizar los umbrales.

Seguidamente se define un umbral entre de 1 y 2. Debido a que el coste de producir una característica siempre es 1, sólo serán generados productos con 1 o 2 características, en este caso, el número de productos válidos es 36, lo cual se muestra en la figura 4.9 (derecha) con un tiempo de ejecución de 0.45 segundos.

Es importante destacar los espacios entre los puntos en la figura 4.9 (derecha). Estos espacios se corresponden a los tiempos requeridos para encontrar la siguiente traza válida (producto). Para generar una traza válida, su coste debe estar entre 1 y 2, mientras que todos los demás productos no serán válidos. Una vez que el coste de producción excede el límite, el cómputo de esa rama termina y no volverá a ser evaluada.

Capítulo 5

SPLA^P: Extensión de SPLA para representar probabilidades

“The probable is what usually happens.”

Aristotle

Contenido

5.1. Sintaxis	124
5.2. Semántica operacional	124
5.3. Consistencia del modelo probabilístico	129
5.4. Semántica denotacional	131
5.5. Equivalencia entre la semántica operacional y la semántica denotacional	137
5.6. Ocultando conjuntos de características	146
5.7. Implementación	149

El objetivo principal de este capítulo es estudiar el comportamiento de las SPLs realizando análisis probabilístico sobre los términos del álgebra. Este análisis muestra que características son las más utilizadas dentro del modelo. Para ello, se debe calcular la probabilidad de encontrar una característica dentro de una línea de productos. La sección 5.1 muestra la descripción formal de la extensión de SPLA para utilizar probabilidades en los elementos sintácticos que lo requieran. Seguidamente, la sección 5.2 muestra los cambios necesarios en las reglas semánticas de SPLA para soportar probabilidades en la semántica operacional del lenguaje. Una vez modificadas, tanto la sintaxis como la semántica de SPLA, la sección 5.3 demuestra que todos los teoremas y lemas planteados para esta extensión son correctos.

La sección 5.4 describe los cambios realizados a las reglas de la semántica denotacional. Seguidamente la sección 5.5 muestra que los productos generados por ambas semánticas (operacional y denotacional) son equivalentes. La sección 5.6 muestra una característica novedosa de la extensión probabilística, la cual consiste en ocultar las características que no afecten a la cardinalidad total de los productos válidos. Finalmente la sección 5.7 presenta la implementación de la semántica denotacional para la extensión probabilística.

5.1. Sintaxis

En el modelo probabilístico propuesto, el conjunto de características será representado como \mathcal{F} , las características aisladas se mostrarán como $A, B, C \dots$, los términos del modelo probabilístico serán descritos como P, Q y las probabilidades p, q son expresadas dentro del rango $(0, 1)$.

Definición 5.1.1 Una *línea de productos probabilística* es un término generado por la siguiente gramática:

$$P ::= \checkmark \mid \text{nil} \mid A; P \mid \bar{A}_{;p} P \mid P \vee_p Q \mid P \wedge Q \mid \\ A \not\Rightarrow B \text{ in } P \mid A \Rightarrow B \text{ in } P \mid P \setminus A \mid P \Rightarrow A$$

donde $A \in \mathcal{F}$, $\checkmark \notin \mathcal{F}$ y $p \in (0, 1)$. Los términos del álgebra serán representados dentro del modelo $\text{SPLA}^{\mathcal{P}}$. \square

Como puede observarse, la descripción de la sintaxis para la extensión probabilística (ver definición 5.1.1) parte de la definición original de SPLA (ver definición 3.1.1). Sin embargo, los elementos sintácticos deben ser modificados para que permitan almacenar información relativa a las probabilidades. Esta información se introduce en los operadores de selección única $P \vee_p Q$ y en el operador de selección opcional $\bar{A}_{;p} P$.

5.2. Semántica operacional

Las reglas de la semántica operacional de $\text{SPLA}^{\mathcal{P}}$ están basadas en las reglas descritas en la figura 3.3. Sin embargo, estas reglas deben ser modificadas y extendidas para permitir

representar probabilidades dentro del nuevo modelo.

Las reglas descritas en la figura 5.1 describen el comportamiento del sistema para procesar términos siguiendo la extensión probabilística SPLA^P . Como se puede observar, la definición de estas reglas utiliza, como base, la definición de la figura 3.3. Sin embargo, las reglas de la extensión probabilística permiten modificar la información relacionada a las probabilidades a medida que se procesan las características.

[tick]	$\checkmark \xrightarrow{1} \text{nil}$	[feat]	$A; P \xrightarrow{A} P$
[ofeat1]	$\bar{A};_p P \xrightarrow{A}_p P$	[ofeat2]	$\bar{A};_p P \xrightarrow{(1-p)} \text{nil}$
[cho1]	$\frac{P \xrightarrow{A}_p P_1}{P \vee_q Q \xrightarrow{A}_{p \cdot q} P_1}$	[cho2]	$\frac{Q \xrightarrow{A}_q Q_1}{P \vee_p Q \xrightarrow{A}_{(1-p) \cdot q} Q_1}$
[con1]	$\frac{P \xrightarrow{A}_p P_1}{P \wedge Q \xrightarrow{A}_{\frac{p}{2}} P_1 \wedge Q}$	[con2]	$\frac{Q \xrightarrow{A}_q Q_1}{P \wedge Q \xrightarrow{A}_{\frac{q}{2}} P \wedge Q_1}$
[con3]	$\frac{P \xrightarrow{\checkmark}_q \text{nil}, Q \xrightarrow{\checkmark}_p \text{nil}}{P \wedge Q \xrightarrow{\checkmark}_{p \cdot q} \text{nil}}$		
[con4]	$\frac{P \xrightarrow{A}_p P_1, Q \xrightarrow{\checkmark}_q \text{nil}}{P \wedge Q \xrightarrow{A}_{\frac{p \cdot q}{2}} P_1}$	[con5]	$\frac{P \xrightarrow{\checkmark}_p \text{nil}, Q \xrightarrow{A}_q Q_1}{P \wedge Q \xrightarrow{A}_{\frac{p \cdot q}{2}} Q_1}$
[req1]	$\frac{P \xrightarrow{C}_p P_1, C \neq A}{A \Rightarrow B \text{ in } P \xrightarrow{C}_p A \Rightarrow B \text{ in } P_1}$	[req2]	$\frac{P \xrightarrow{A}_p P_1}{A \Rightarrow B \text{ in } P \xrightarrow{A}_p P_1 \Rightarrow B}$
[req3]	$\frac{P \xrightarrow{\checkmark}_p \text{nil}}{A \Rightarrow B \text{ in } P \xrightarrow{\checkmark}_p \text{nil}}$		
[excl1]	$\frac{P \xrightarrow{C}_p P_1, C \neq A, C \neq B}{A \not\Rightarrow B \text{ in } P \xrightarrow{C}_p A \not\Rightarrow B \text{ in } P_1}$	[excl2]	$\frac{P \xrightarrow{A}_p P_1}{A \not\Rightarrow B \text{ in } P \xrightarrow{A}_p P_1 \setminus B}$
[excl3]	$\frac{P \xrightarrow{B}_p P_1}{A \not\Rightarrow B \text{ in } P \xrightarrow{B}_p P_1 \setminus A}$	[excl4]	$\frac{P \xrightarrow{\checkmark}_p \text{nil}}{A \not\Rightarrow B \text{ in } P \xrightarrow{\checkmark}_p \text{nil}}$
[forb1]	$\frac{P \xrightarrow{B}_p P_1, B \neq A}{P \setminus A \xrightarrow{B}_p P_1 \setminus A}$	[forb2]	$\frac{P \xrightarrow{\checkmark}_p \text{nil}}{P \setminus A \xrightarrow{\checkmark}_p \text{nil}}$
[mand1]	$\frac{P \xrightarrow{\checkmark}_p \text{nil}}{P \Rightarrow A \xrightarrow{A}_p \checkmark}$	[mand2]	$\frac{P \xrightarrow{A}_p P_1}{P \Rightarrow A \xrightarrow{A}_p P_1}$
[mand3]	$\frac{P \xrightarrow{B}_p P_1, A \neq B}{P \Rightarrow A \xrightarrow{B}_p P_1 \Rightarrow A}$		

$A, B, C \in \mathcal{F}, a \in \mathcal{F} \cup \{\checkmark\}$

Figura 5.1: Reglas para definir la semántica operacional de SPLA .

Las reglas **[tick]** y **[feat]** muestran que procesar las características \checkmark y **A**, respectivamente, podrá realizarse siempre con probabilidad 1. Las transiciones **[ofeat1]** y **[ofeat2]** permiten procesar el operador opcional, esto es, si se procesa la característica, se computa con probabilidad p , En caso contrario, se computa con probabilidad $1 - p$.

El operador de selección única (\vee) está compuesto por dos reglas semánticas, **[cho1]** y **[cho2]**. En el caso de la regla **[cho1]**, se selecciona la característica con probabilidad p . Para la regla **[cho2]**, al ser un operador simétrico, la probabilidad es $(1 - p) \cdot q$.

Los operadores de conjunción **[con1]** y **[con2]** procesan el operador de paralelo siempre que sea posible continuar procesando características. Para ello, la probabilidad de procesar cada parte de la conjunción es $\frac{p}{2}$ o $\frac{q}{2}$, dependiendo de la parte que se esté ejecutando. La regla **[con3]** permite procesar la característica \checkmark cuando ésta pueda ser procesada en ambos lados del paralelo. Su probabilidad será la multiplicación de las probabilidades de ambos lados ($p \cdot q$). Las reglas **[con4]** y **[con5]** permiten procesar el operador de conjunción si alguno de los extremos del término puede generar la característica \checkmark . Su probabilidad será la mitad de la multiplicación de las probabilidades de ambos lados del operador, esto es, $\frac{p \cdot q}{2}$.

Todos los operadores restantes no intervienen ni modifican la probabilidad de los elementos sintácticos al ser procesados.

Se utilizan *multiconjuntos* con el objetivo de considerar distintas ocurrencias de la misma transición. De esta manera, si una transición puede ser derivada de distintas formas, entonces cada derivación genera una instancia distinta de esta transición. Por ejemplo, consideremos el término $P = \bar{A} \vee_{\frac{1}{2}} \bar{A}$, donde las apariciones finales de \checkmark han sido omitidas. Si no se es cuidadoso al procesar los términos se podría obtener la transición $\bar{A} \vee_{\frac{1}{2}} \bar{A} \xrightarrow{A} \frac{1}{2} \text{ nil}$ una sola vez. Sin embargo, esta transición se debe obtener dos veces. De esta manera, si una transición puede ser derivada de distintas formas, consideramos que cada derivación genera una instancia diferente. En particular, se utilizarán los delimitadores \lfloor y \rfloor para describir multiconjuntos y \uplus para mostrar la unión de multiconjuntos.

La demostración del siguiente resultado es inmediata, la cual describe que una transición

satisfactoria termina con el símbolo **nil**.

Lema 5.2.1 Dados los términos $P, Q \in \mathbf{SPLA}^{\mathcal{P}}$ y la probabilidad $p \in (0, 1]$. Se tiene que $P \xrightarrow{\checkmark}_p Q$, si y solo si $Q = \mathbf{nil}$. \square

El comportamiento del símbolo terminal debe ser previamente definido para generar un producto válido de la extensión probabilística. El lema 5.2.1 muestra que sólo es posible procesar la característica \checkmark cuando el término restante es **nil**, lo que significa que ha sido generado un producto válido.

Una vez definido el modelado de la información probabilística en la semántica operacional de $\mathbf{SPLA}^{\mathcal{P}}$, debe describirse el proceso del cálculo de las probabilidades. Básicamente, se consigue mediante la definición de las funciones que permitirán calcular la probabilidad de una traza probabilística, dentro de un término que pertenezca al álgebra. Seguidamente, debe definirse el concepto de trazas probabilísticas, la probabilidad de cualquier producto y el cálculo de su desecho.

A continuación, se describen los lemas relacionados con la composición de transiciones consecutivas.

Definición 5.2.2 Dados los términos $P, Q \in \mathbf{SPLA}^{\mathcal{P}}$, se escribe $P \xRightarrow{s}_p Q$ si existe una secuencia consecutiva de transiciones

$$P = P_0 \xrightarrow{a_1}_{p_1} P_1 \xrightarrow{a_2}_{p_2} P_2 \cdots P_{n-1} \xrightarrow{a_n}_{p_n} P_n = Q$$

donde $n \geq 0$, $s = a_1 a_2 \cdots a_n$ y $p = p_1 \cdot p_2 \cdots p_n$. Se dice que s es una traza de P .

Recordemos que la traza $s \in \mathcal{F}^*$. El producto $\lfloor s \rfloor \subseteq \mathcal{F}$ es el que contiene, como conjunto, todas las características pertenecientes a s .

Dado el término $P \in \mathbf{SPLA}^{\mathcal{P}}$, se define el conjunto de productos probabilísticos de P , denotado por $\mathbf{prod}^{\mathcal{P}}(P)$, como el conjunto

$$\{(pr, p) \mid p > 0 \wedge p = \sum q \mid P \xRightarrow{s\checkmark}_q Q \wedge \lfloor s \rfloor = pr\}$$

Se define la probabilidad total de P , denotado por $\text{TotProb}(P)$, como el valor:

$$\sum \{p \mid \exists pr : (pr, p) \in \text{prod}^P(P)\}$$

Adicionalmente, se define $\text{waste}(P) = 1 - \text{TotProb}(P)$. El desecho de P se define como la probabilidad residual de $\text{TotProb}(P)$, es decir, el sumatorio de las probabilidades de los productos no válidos. \square

El siguiente resultado muestra algunas propiedades relacionadas con las probabilidades dentro de la semántica operacional. En particular, se muestra que la probabilidad de una transición, o de una secuencia de transiciones, es mayor que cero y que las probabilidades de los productos están en el rango $[0, 1]$.

Definición 5.2.3 Dados los términos $P, Q \in \text{SPLA}^P$ y la característica $A \in \mathcal{F} \cup \{\checkmark\}$, existe una transición de P a Q , etiquetada con el símbolo A y con la probabilidad p , denotada por $P \xrightarrow{A}_p Q$, si puede ser deducida de las reglas descritas en la figura 5.1. \square

Para poder definir la semántica de nuestro lenguaje probabilístico, es necesario presentar el concepto de multiconjunto, el cual consiste en un conjunto en el que podemos tener elementos repetidos. Formalmente, un multiconjunto M de A es una función

$$M : A \mapsto \mathbb{N}$$

que indica el número de repeticiones de cada elemento. Si $M(a) = 0$ diremos que a no pertenece a M , que se denota por $a \notin M$. Si $M(a) > 0$, diremos que a pertenece a M , denotado por $a \in M$.

Las definiciones 5.2.2 y 5.2.3 muestran cómo se modela la información probabilística dentro de SPLA^P . Esta información es relativa al procesamiento de las características que pertenezcan a cualquier término válido y a trazas de características.

Lema 5.2.4 Dados los términos $P, Q \in \text{SPLA}^P$, se obtienen los siguientes resultados.

1. Si $P \xrightarrow{A}_p Q$ entonces $p \in (0, 1]$. La demostración es trivial tras una inspección de las reglas de la semántica operacional.
2. Si $P \xRightarrow{s}_p Q$ entonces $p \in (0, 1]$. La demostración es consecuencia de la primera y se demuestra por inducción sobre la longitud de la traza.
3. $\sum \{p \mid \exists A \in \mathcal{F}, Q \in \text{SPLA}^{\mathcal{P}} : P \xrightarrow{A}_p Q\} \in [0, 1]$.
4. $\sum \{p \mid \exists s \in \mathcal{F}^*, Q \in \text{SPLA}^{\mathcal{P}} : P \xRightarrow{s\checkmark}_p Q\} \in [0, 1]$.
5. $\text{TotProb}(P) \in [0, 1]$.

□

Así, el lema 5.2.4 acota el sumatorio de todas las probabilidades del procesamiento de las características, representado por números reales, al intervalo $[0, 1]$.

Lema 5.2.5 Dado el término $P \in \text{SPLA}^{\mathcal{P}}$.

1. Si $(pr, p) \in \text{prod}^{\mathcal{P}}(P)$, entonces $p \in (0, 1]$.
2. $0 \leq \text{TotProb}(P) \leq 1$.

Demostración: Este lema se plantea como una consecuencia directa del lema 5.2.4, □

El lema 5.2.5 hace referencia a la suma de las probabilidades, el cual acota la probabilidad de un producto válido $(0, 1]$ y el sumatorio de todos los productos válidos $[0, 1]$.

Los ejemplos descritos en la figura 5.2 muestran el cálculo de la información probabilística al procesar los términos P_1 , P_2 y P_3 . La ejecución de las reglas de la semántica operacional se pueden observar en estos tres ejemplos.

5.3. Consistencia del modelo probabilístico

La consistencia del modelo probabilístico se basa en la consistencia del modelo no probabilístico. Esto implica que se deberá definir una función que permita realizar una traducción directa entre $\text{SPLA}^{\mathcal{P}}$ y SPLA . De esta manera, se demostrará que el modelo es consistente, si

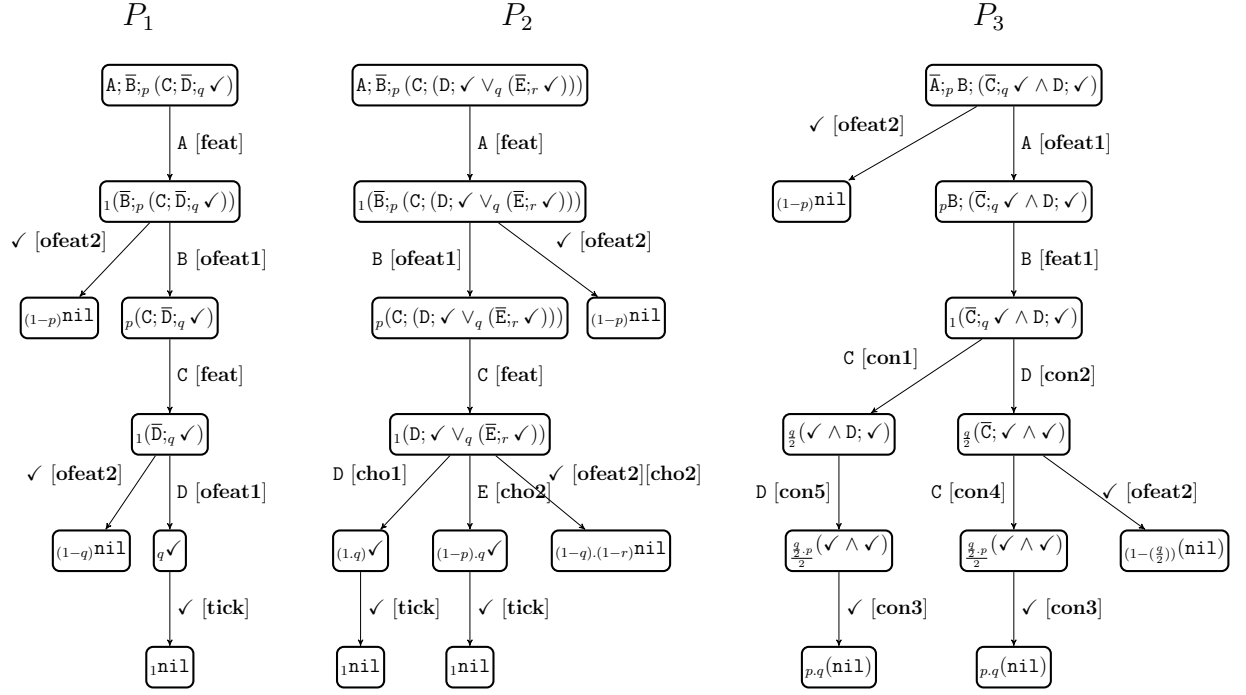


Figura 5.2: Ejemplos de la ejecución de la reglas de la semántica operacional del modelo probabilístico.

y sólo si el modelo no probabilístico lo es. La consistencia es una propiedad importante en el lenguaje propuesto. Decimos que el modelo no probabilístico es *consistente*, si es capaz de generar productos [4]. En este caso se define la consistencia teniendo $\text{TotProb}(P) > 0$. Además se demuestra que una traducción del modelo probabilístico al no probabilístico mantiene su consistencia de la manera esperada.

La sintaxis del modelo no probabilístico se describe en detalle en la definición 3.1.1.

Definición 5.3.1 Se define una función de traducción $\text{np} : \text{SPLA}^P \mapsto \text{SPLA}$ de la siguiente manera:

$$\text{np}(P) = \begin{cases} \checkmark & \text{if } P = \checkmark \\ \text{nil} & \text{if } P = \text{nil} \\ A; \text{np}(P) & \text{si } P = A; P \\ \bar{A}; \text{np}(P) & \text{si } P = \bar{A};_p P \\ \text{np}(P) \vee \text{np}(Q) & \text{si } P \vee_p Q \\ \text{np}(P) \wedge \text{np}(Q) & \text{si } P \wedge Q \\ A \Rightarrow B \text{ in } \text{np}(P) & \text{si } A \Rightarrow B \text{ in } P \\ A \not\Rightarrow B \text{ in } \text{np}(P) & \text{si } A \not\Rightarrow B \text{ in } P \\ \text{np}(P) \Rightarrow A & \text{si } P \Rightarrow A \\ \text{np}(P) \setminus A & \text{si } P \setminus A \end{cases}$$

□

La demostración del siguiente resultado es directa al tener en cuenta que las reglas de la semántica operacional son las mismas para ambos modelos. Siempre que las probabilidades sean eliminadas como se describe en [4]. Cualquier secuencia de transiciones derivadas en el modelo probabilístico puede ser derivada del modelo no probabilístico. Adicionalmente, por el lema 5.2.4, se conoce que cualquier traza derivada del modelo probabilístico tiene la propiedad de que no es nula.

Teorema 5.3.2 Dados los términos $P, Q \in \text{SPLA}^P$, se tiene $P \xRightarrow{s}_p Q$, si y solo si $\text{np}(P) \xRightarrow{s} \text{np}(Q)$. Además, $pr \in \text{prod}(\text{np}(P))$, si y solo si existe $p > 0$ tal que $(pr, p) \in \text{prod}^P(P)$. □

5.4. Semántica denotacional

La semántica de cualquier expresión SPLA^P se representa por el conjunto de sus productos y sus probabilidades. Cada producto puede ser representado por sus características sin tener en cuenta su posición dentro de la traza, de tal manera se muestran como conjuntos de características. De forma similar a como se describe en la sección 3.4, la extensión denotacional de SPLA^P permitirá definir todas las funciones que describen el comportamiento del modelo sin importar la forma en que las reglas son procesadas. La semántica denotacional

se basa en las definiciones del modelo no probabilístico, con la excepción de que debe ser añadida la información probabilística a medida que se procesan las características.

La forma de definir la semántica denotacional para la extensión probabilística, comienza definiendo el *dominio matemático* donde serán representados los elementos sintácticos de $\text{SPLA}^{\mathcal{P}}$.

Definición 5.4.1 Se define el dominio semántico \mathcal{M} como el conjunto más grande $\mathcal{M} \subseteq \mathcal{P}(\mathcal{P}(\mathcal{F}) \times (0, 1])$, de tal manera que $A \in \mathcal{M}$, entonces se mantienen las siguientes condiciones:

1. Si $(P, q) \in A$ y $(P, r) \in A$, entonces $q = r$.
2. $0 \leq \sum \{q \mid \exists P : (P, q) \in A\} \leq 1$.

Para la descripción de la semántica denotacional se utilizarán multiconjuntos y estos estarán representados por $\{ \cdots \}$.

Dado el multiconjunto M con elementos dentro del conjunto $\mathcal{P}(\mathcal{F}) \times [0, 1]$, se define el operador **accum** de la siguiente manera:

$$\text{accum}(M) = \left\{ (P, p) \mid p = \sum_{(P, q) \in M} q \wedge p > 0 \right\}$$

□

Aunque los elementos del dominio semántico estén compuestos por conjuntos de pares (producto, probabilidad) con hasta una ocurrencia de un producto dado, se utilizarán los multiconjuntos como elementos auxiliares dentro de las funciones semánticas. De esta forma, la función **accum**(M) será utilizada para convertir estos multiconjuntos en conjuntos. El siguiente resultado es inmediato.

Proposición 5.4.2 Dado el multiconjunto M con elementos en el conjunto $\mathcal{P}(\mathcal{F}) \times [0, 1]$, si $1 \geq \sum \{q \mid (P, q) \in M\}$, entonces **accum**(M) $\in \mathcal{M}$. □

A continuación se definen los operadores de la semántica denotacional. Como se ha explicado anteriormente, los multiconjuntos que cumplen las condiciones del resultado previo surgen al definir estos operadores. Por ejemplo, el operador de prefijo $\llbracket \mathbf{A}; \rrbracket(M)$ debe agregar la característica \mathbf{A} a cualquier producto en M . Suponemos que $M = \{(\{\mathbf{B}, \mathbf{A}\}, \frac{1}{2}), (\{\mathbf{B}\}, \frac{1}{2})\}$. Si es \mathbf{A} añadido a los productos de M , entonces obtenemos el producto $\{\mathbf{A}, \mathbf{B}\}$ duplicado con una probabilidad $\frac{1}{2}$ asociada a cada ocurrencia. Por ello, es necesario aplicar la función **accum** para sumar ambas probabilidades y obtener así un único producto con probabilidad 1.

El siguiente paso consiste en definir un operador semántico para cada uno de los operadores sintácticos en **SPLA**, lo cual se lleva a cabo mediante la siguiente definición.

Definición 5.4.3 Los operadores de la semántica denotacional para la extensión probabilística de **SPLA** se definen como:

- $\llbracket \mathbf{nil} \rrbracket^{\mathcal{P}} = \emptyset$

La función de la semántica denotacional que recibe **nil** retornará siempre vacío. Así ninguna otra característica podrá ser procesada, ya que no existe ninguna función definida que reciba \emptyset como parámetro.

- $\llbracket \checkmark \rrbracket^{\mathcal{P}} = \{(\emptyset, 1)\}$

La función que recibe \checkmark como parámetro siempre será procesada con probabilidad 1, ya que \checkmark indica que se ha generado un producto válido.

- $\llbracket \mathbf{A}; \cdot \rrbracket^{\mathcal{P}} : \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \mathbf{A}; \cdot \rrbracket^{\mathcal{P}}(M) = \mathbf{accum}(\lambda(\{\mathbf{A}\} \cup P, p) \mid (P, p) \in M)$$

Cualquier característica obligatoria será procesada con probabilidad p . Además, ésta será añadida al conjunto de características procesadas previamente.

- $\llbracket \bar{\mathbf{A}};_r \cdot \rrbracket^{\mathcal{P}} : \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \bar{\mathbf{A}};_r \cdot \rrbracket^{\mathcal{P}}(M) = \mathbf{accum}(\lambda(\emptyset, 1 - r) \int \uplus \lambda(\{\mathbf{A}\} \cup P, r \cdot p) \mid (P, p) \in M)$$

Al procesar una característica opcional deben realizarse dos acciones. La primera acción consiste en no procesar la característica, terminando el procesamiento del término con probabilidad $1 - r$. La segunda consiste en procesarla con probabilidad r , siendo ésta multiplicada por la probabilidad acumulada del término restante.

- $\llbracket \cdot \vee_r \cdot \rrbracket^{\mathcal{P}} : \mathcal{M} \times \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \cdot \vee_r \cdot \rrbracket^{\mathcal{P}}(M_1, M_2) = \text{accum}(\lambda(P, r \cdot p) \mid (P, p) \in M_1 \mathbin{\mathcal{J}} \uplus \lambda(Q, (1 - r) \cdot q) \mid (Q, q) \in M_2 \mathbin{\mathcal{J}})$$

El operador de selección única separa los términos a ambos lados del operador. Para ambos lados, el cálculo de la probabilidad será igual a multiplicar sus probabilidades. Es importante tomar en cuenta que existirán dos casos. El primer caso, que se procese la transición con probabilidad r o el segundo caso, que la transición no se procese con probabilidad $1 - r$.

- $\llbracket \cdot \wedge \cdot \rrbracket^{\mathcal{P}} : \mathcal{M} \times \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \cdot \wedge \cdot \rrbracket^{\mathcal{P}}(M_1, M_2) = \text{accum}(\lambda(P \cup Q, p \cdot q) \mid (P, p) \in M_1, (Q, q) \in M_2 \mathbin{\mathcal{J}})$$

El operador de paralelo, dentro de la semántica denotacional, unirá las características de ambos términos multiplicando su probabilidad.

- $\llbracket \mathbf{A} \Rightarrow \mathbf{B} \text{ in } \cdot \rrbracket^{\mathcal{P}} : \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \mathbf{A} \Rightarrow \mathbf{B} \text{ in } \cdot \rrbracket^{\mathcal{P}}(M) = \text{accum} \left(\begin{array}{l} \lambda(P, p) \mid (P, p) \in M, \mathbf{A} \notin P \mathbin{\mathcal{J}} \uplus \\ \lambda(\{\mathbf{B}\} \cup P, p) \mid (P, p) \in M, \mathbf{A} \in P \mathbin{\mathcal{J}} \end{array} \right)$$

La restricción de requerimiento no modifica las probabilidades de los productos en construcción. Sin embargo, existen dos casos. El primer caso, consiste en que no se cumpla la restricción de requerimiento y el segundo caso consiste en que esta se cumpla.

- $\llbracket \mathbf{A} \not\Rightarrow \mathbf{B} \text{ in } \cdot \rrbracket^{\mathcal{P}} : \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \mathbf{A} \not\Rightarrow \mathbf{B} \text{ in } \cdot \rrbracket^{\mathcal{P}}(M) = \{(P, p) \mid (P, p) \in M, \mathbf{A} \notin P\} \cup \{(P, p) \mid (P, p) \in M, \mathbf{B} \notin P\}$$

Para el operador de la restricción de exclusión, se procesara el término en dos partes, de tal manera que las características A y B no coincidan en un mismo producto.

- $\llbracket \cdot \Rightarrow A \rrbracket^{\mathcal{P}} : \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \cdot \Rightarrow A \rrbracket^{\mathcal{P}}(M) = \llbracket A; \cdot \rrbracket^{\mathcal{P}}(M)$$

Esta regla obliga a que la característica referida por el operador de inclusión se agregue inmediatamente al término.

- $\llbracket \cdot \setminus A \rrbracket^{\mathcal{P}} : \mathcal{M} \mapsto \mathcal{M}$ como

$$\llbracket \cdot \setminus A \rrbracket^{\mathcal{P}}(M) = \{(P, p) \mid (P, p) \in M, A \notin P\}$$

Esta regla obliga a que la característica referida por el operador de ocultamiento sea eliminada del término, antes de generar el producto válido.

□

Resulta sencillo verificar que todos los multiconjuntos que han aparecido en la definición previa cumplen las condiciones de la proposición 5.4.2. De esta manera, los operadores están bien definidos. Esta condición se formaliza en el siguiente resultado.

Proposición 5.4.4 Dados los multiconjuntos $M, M_1, M_2 \in \mathcal{M}$, la probabilidad $p \in (0, 1]$ y las características $A, B \in \mathcal{F}$, entonces:

- | | |
|--|--|
| ▪ $\llbracket A; \cdot \rrbracket^{\mathcal{P}}(M) \in \mathcal{M}$ | ▪ $\llbracket A \Rightarrow B \text{ in } \cdot \rrbracket^{\mathcal{P}}(M) \in \mathcal{M}$ |
| ▪ $\llbracket \bar{A}; \cdot \rrbracket^{\mathcal{P}}(M) \in \mathcal{M}$ | ▪ $\llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket^{\mathcal{P}}(M) \in \mathcal{M}$ |
| ▪ $\llbracket \cdot \vee_r \cdot \rrbracket^{\mathcal{P}}(M_1, M_2) \in \mathcal{M}$ | ▪ $\llbracket \cdot \Rightarrow A \rrbracket^{\mathcal{P}}(M) \in \mathcal{M}$ |
| ▪ $\llbracket \cdot \wedge \cdot \rrbracket^{\mathcal{P}}(M_1, M_2) \in \mathcal{M}$ | ▪ $\llbracket \cdot \setminus A \rrbracket^{\mathcal{P}}(M) \in \mathcal{M}$ |

Demostración. La demostración de la proposición 5.4.4 permite comprobar que la ejecución de las reglas de la semántica denotacional de SPLA^P siempre pertenecen a \mathcal{M} . Esto es una consecuencia directa de la proposición 5.4.2. \square

Una vez definidos los operadores semánticos sobre un conjunto de productos, es posible definir la semántica denotacional para cualquier expresión SPLA . Ésta se define de manera inductiva.

$$\begin{aligned}
P_1 &= \mathbf{A}; \bar{\mathbf{B}}_p \checkmark \\
\llbracket \checkmark \rrbracket^P &= \{(\emptyset, 1)\} \\
\llbracket \bar{\mathbf{B}}_p \checkmark \rrbracket^P &= \llbracket \bar{\mathbf{B}}_p \cdot \rrbracket^P(\llbracket \checkmark \rrbracket^P) = \llbracket \bar{\mathbf{B}}_p \cdot \rrbracket^P(\{(\emptyset, 1)\}) = \\
&\quad \{(\emptyset, (1-p)), (\mathbf{B}, p)\} \\
\llbracket \mathbf{A}; \bar{\mathbf{B}}_p \checkmark \rrbracket^P &= \llbracket \mathbf{A}; \cdot \rrbracket^P(\llbracket \bar{\mathbf{B}}_p \checkmark \rrbracket^P) = \llbracket \mathbf{A}; \cdot \rrbracket^P(\{(\emptyset, (1-p)), (\mathbf{B}, p)\}) = \\
&\quad \{(\mathbf{A}, (1-p)), (\mathbf{AB}, p)\}
\end{aligned}$$

$$\begin{aligned}
P_2 &= \mathbf{A}; (\bar{\mathbf{B}}_p \checkmark \vee_q \bar{\mathbf{B}}_r \checkmark) \\
\llbracket \checkmark \rrbracket^P &= \{(\emptyset, 1)\} \\
\llbracket \bar{\mathbf{B}}_p \checkmark \rrbracket^P &= \llbracket \bar{\mathbf{B}}_p \cdot \rrbracket^P(\llbracket \checkmark \rrbracket^P) = \llbracket \bar{\mathbf{B}}_p \cdot \rrbracket^P(\{(\emptyset, 1)\}) = \\
&\quad \{(\emptyset, (1-p)), (\mathbf{B}, p)\} \\
\llbracket \bar{\mathbf{B}}_r \checkmark \rrbracket^P &= \llbracket \bar{\mathbf{B}}_r \cdot \rrbracket^P(\llbracket \checkmark \rrbracket^P) = \llbracket \bar{\mathbf{B}}_r \cdot \rrbracket^P(\{(\emptyset, 1)\}) = \\
&\quad \{(\emptyset, (1-r)), (\mathbf{B}, r)\} \\
\llbracket \bar{\mathbf{B}}_p \checkmark \vee_q \bar{\mathbf{B}}_r \checkmark \rrbracket^P &= \llbracket \cdot \vee_q \cdot \rrbracket^P(\llbracket \bar{\mathbf{B}}_p \checkmark \rrbracket^P, \llbracket \bar{\mathbf{B}}_r \checkmark \rrbracket^P) \\
&= \llbracket \cdot \vee_q \cdot \rrbracket^P(\{(\emptyset, (1-p)), (\mathbf{B}, p)\}, \{(\emptyset, (1-r)), (\mathbf{B}, r)\}) = \\
&= \text{accum}\left(\{(\emptyset, (q) \cdot (1-p)), (\mathbf{B}, q \cdot p)\} \uplus \{(\emptyset, (1-q) \cdot (1-r)), (\mathbf{B}, (1-q) \cdot r)\}\right) = \\
&= \{(\emptyset, ((q) \cdot (1-p)) + ((1-q) \cdot (1-r))), (\mathbf{B}, (q \cdot p) + ((1-q) \cdot r))\}
\end{aligned}$$

$$\begin{aligned}
P_3 &= \mathbf{A}; (\mathbf{A}; \checkmark \vee_q \mathbf{B}; \checkmark) \\
\llbracket \checkmark \rrbracket^P &= \{(\emptyset, 1)\} \\
\llbracket \mathbf{A}; \checkmark \rrbracket^P &= \llbracket \mathbf{A}; \cdot \rrbracket^P(\llbracket \checkmark \rrbracket^P) = \llbracket \mathbf{A}; \cdot \rrbracket^P(\{(\emptyset, 1)\}) = \{(\mathbf{A}, 1)\} \\
\llbracket \mathbf{A}; \checkmark \vee_q \mathbf{B}; \checkmark \rrbracket^P &= \llbracket \cdot \vee_q \cdot \rrbracket^P(\llbracket \mathbf{A}; \checkmark \rrbracket^P, \llbracket \mathbf{B}; \checkmark \rrbracket^P) = \\
&= \llbracket \cdot \vee_q \cdot \rrbracket^P(\{(\mathbf{A}, 1)\}, \{(\mathbf{B}, 1)\}) = \{(\mathbf{A}, q), (\mathbf{B}, (1-q))\} \\
\llbracket \mathbf{A}; (\mathbf{A}; \checkmark \vee_q \mathbf{B}; \checkmark) \rrbracket^P &= \llbracket \mathbf{A}; \cdot \rrbracket^P(\llbracket \mathbf{A}; \checkmark \vee_q \mathbf{B}; \checkmark \rrbracket^P) = \\
&= \llbracket \mathbf{A}; \cdot \rrbracket^P(\{(\mathbf{A}, q), (\mathbf{B}, (1-q))\}) = \{(\mathbf{A}, q), (\mathbf{AB}, (1-q))\}
\end{aligned}$$

Figura 5.3: Ejemplo de la ejecución de las reglas de la semántica denotacional (1 de 2).

Definición 5.4.5 La semántica denotacional de SPLA se representa por la función:

$$\llbracket \cdot \rrbracket^P : \text{SPLA} \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{F}))$$

Esta función se define de manera inductiva de la siguiente forma. Para cualquier operador n -ario $\text{op} \in \{\text{nil}, \checkmark, \mathbf{A}; \cdot, \bar{\mathbf{A}}; \cdot, \cdot \vee_p \cdot, \cdot \wedge \cdot, \mathbf{A} \Rightarrow \mathbf{B} \text{ in } \cdot, \mathbf{A} \not\Rightarrow \mathbf{B} \text{ in } \cdot, \cdot \Rightarrow \mathbf{A}, \cdot \setminus \mathbf{A}\}^1$:

$$\llbracket \text{op}(P_1, \dots, P_n) \rrbracket^{\mathcal{P}} = \llbracket \text{op} \rrbracket^{\mathcal{P}}(\llbracket P_1 \rrbracket^{\mathcal{P}}, \dots, \llbracket P_n \rrbracket^{\mathcal{P}})$$

□

$$\begin{aligned}
P_4 &= (\mathbf{A}; \checkmark \vee_p \mathbf{B}; \text{nil}) \\
\llbracket \text{nil} \rrbracket^{\mathcal{P}} &= \emptyset \\
\llbracket \mathbf{B}; \text{nil} \rrbracket^{\mathcal{P}} &= \llbracket \mathbf{B}; \cdot \rrbracket^{\mathcal{P}}(\llbracket \text{nil} \rrbracket^{\mathcal{P}}) = \llbracket \mathbf{B}; \cdot \rrbracket^{\mathcal{P}}(\emptyset) = \emptyset \\
\llbracket \checkmark \rrbracket^{\mathcal{P}} &= \{(\emptyset, 1)\} \\
\llbracket \mathbf{A}; \checkmark \rrbracket^{\mathcal{P}} &= \llbracket \mathbf{A}; \cdot \rrbracket^{\mathcal{P}}(\llbracket \checkmark \rrbracket^{\mathcal{P}}) = \llbracket \mathbf{A}; \cdot \rrbracket^{\mathcal{P}}(\{(\emptyset, 1)\}) = \{(\mathbf{A}, 1)\} \\
\llbracket \mathbf{A}; \checkmark \vee_p \mathbf{B}; \text{nil} \rrbracket^{\mathcal{P}} &= \llbracket \cdot \vee_p \cdot \rrbracket^{\mathcal{P}}(\llbracket \mathbf{A}; \checkmark \rrbracket^{\mathcal{P}}, \llbracket \mathbf{B}; \text{nil} \rrbracket^{\mathcal{P}}) \\
&= \llbracket \cdot \vee_p \cdot \rrbracket^{\mathcal{P}}(\{(\mathbf{A}, 1)\}, \emptyset) = \{(\mathbf{A}, 1)\}
\end{aligned}$$

$$\begin{aligned}
P_5 &= (\mathbf{A}; \checkmark \wedge \mathbf{B}; \checkmark) \\
\llbracket \checkmark \rrbracket^{\mathcal{P}} &= \{(\emptyset, 1)\} \\
\llbracket \mathbf{A}; \checkmark \rrbracket^{\mathcal{P}} &= \llbracket \mathbf{A}; \cdot \rrbracket^{\mathcal{P}}(\llbracket \checkmark \rrbracket^{\mathcal{P}}) = \llbracket \mathbf{A}; \cdot \rrbracket^{\mathcal{P}}(\{(\emptyset, 1)\}) = \{(\mathbf{A}, 1)\} \\
\llbracket \mathbf{A}; \checkmark \wedge \mathbf{B}; \checkmark \rrbracket^{\mathcal{P}} &= \llbracket \cdot \wedge \cdot \rrbracket^{\mathcal{P}}(\llbracket \mathbf{A}; \checkmark \rrbracket^{\mathcal{P}}, \llbracket \mathbf{B}; \checkmark \rrbracket^{\mathcal{P}}) = \llbracket \cdot \wedge \cdot \rrbracket^{\mathcal{P}}(\{(\mathbf{A}, 1)\}, \{(\mathbf{B}, 1)\}) = \\
&\quad \{(\mathbf{AB}, \tfrac{1}{2}), (\mathbf{BA}, \tfrac{1}{2})\}
\end{aligned}$$

Figura 5.4: Ejemplo de la ejecución de las reglas de la semántica denotacional (2 de 2).

En la figura 5.3 y en la figura 5.4 se ilustra el cálculo la semántica denotacional de 5 ejemplos. En cada uno de estos ejemplos, se analiza el proceso de cálculo de las probabilidades de las características dentro de términos a medida que son procesados. El término se representa como un árbol, desde los elementos terminales u hojas hasta la raíz.

5.5. Equivalencia entre la semántica operacional y la semántica denotacional

Esta sección muestra las relaciones de equivalencia entre la semántica operacional y la semántica denotacional.

¹ nil y \checkmark son operadores 0-arios; $\mathbf{A}; \cdot$, $\bar{\mathbf{A}}; \cdot$, $\mathbf{A} \Rightarrow \mathbf{B} \text{ in } \cdot$, $\mathbf{A} \not\Rightarrow \mathbf{B} \text{ in } \cdot$, $\cdot \Rightarrow \mathbf{A}$, $\cdot \setminus \mathbf{A}$ son operadores 1-arios; $\cdot \vee_p \cdot$ y $\cdot \wedge \cdot$ son operadores 2-arios.

Para la extensión probabilística se han definido dos semánticas, la semántica operacional y la semántica denotacional, de las cuales pueden derivarse productos válidos. Es importante que ambas semánticas coincidan, de forma que pueda escogerse cualquiera de ellas dependiendo del problema que sea necesario resolver. La demostración del siguiente resultado es una consecuencia inmediata de los lemas 5.5.3 al 5.5.12.

Proposición 5.5.1 Dados los términos $P, Q \in \text{SPLA}^P$, las características $A, B \in \mathcal{F}$ y la probabilidad $q \in (0, 1)$, se obtienen los siguientes resultados:

$$\begin{aligned}
\text{prod}^P(A; P) &= \llbracket A; \cdot \rrbracket^P(\text{prod}^P(P)) \\
\text{prod}^P(\bar{A};_q P) &= \llbracket \bar{A};_q \cdot \rrbracket^P(\text{prod}^P(P)) \\
\text{prod}^P(P \vee_q Q) &= \llbracket \cdot \vee_q \cdot \rrbracket^P(\text{prod}^P(P), \text{prod}^P(Q)) \\
\text{prod}^P(P \wedge Q) &= \llbracket \cdot \wedge \cdot \rrbracket^P(\text{prod}^P(P), \text{prod}^P(Q)) \\
\text{prod}^P(P \Rightarrow A) &= \llbracket \cdot \Rightarrow A \rrbracket^P(\text{prod}^P(P)) \\
\text{prod}^P(P \setminus A) &= \llbracket \cdot \setminus A \rrbracket^P(\text{prod}^P(P)) \\
\text{prod}^P(A \Rightarrow B \text{ in } P) &= \llbracket A \Rightarrow B \text{ in } \cdot \rrbracket^P(\text{prod}^P(P)) \\
\text{prod}^P(A \not\Rightarrow B \text{ in } P) &= \llbracket A \not\Rightarrow B \text{ in } \cdot \rrbracket^P(\text{prod}^P(P))
\end{aligned}$$

□

La demostración por inducción estructural de P , es sencilla mediante el uso de la proposición 5.5.1.

Teorema 5.5.2 Dado el término $P \in \text{SPLA}^P$, el producto $pr \subseteq \mathcal{F}$ y la probabilidad $p \in (0, 1]$, se obtiene que $(pr, p) \in \llbracket P \rrbracket^P$, si y solo si $(pr, p) \in \text{prod}^P(P)$. □

Lema 5.5.3 Dado el término $P \in \text{SPLA}^P$ y la característica $A \in \mathcal{F}$, entonces $(pr, p) \in \text{prod}^P(A; P)$, si y solo si

$$p = \sum \{r \mid (pr', r) \in \text{prod}^P(P) \wedge pr' \cup \{A\} = pr\}$$

Demostración: La otra transición de $A; P$ es $A; P \xrightarrow{A}_1 Q$. Entonces $A; P \xRightarrow{s}_p P$, si y solo si

$$A; P \xrightarrow{A}_1 P \xRightarrow{s}_p Q \quad \wedge \quad s = A \cdot s'$$

entonces

$$\begin{aligned}
p &= \sum \wr r \mid \mathbf{A}; P \xRightarrow{s\checkmark}_p \mathbf{nil} \wedge \lfloor s \rfloor = pr \wr = \\
&\sum \wr r \mid \mathbf{A}; P \xrightarrow{\mathbf{A}}_1 P \xRightarrow{s'\checkmark}_r \mathbf{nil} \wedge \lfloor \mathbf{A} \cdot s' \rfloor = pr \wr \\
&\quad \sum \wr r \mid P \xRightarrow{s'\checkmark}_r \mathbf{nil} \wedge \{\mathbf{A}\} \cup \lfloor s' \rfloor = pr \wr \\
&\quad \sum \wr r \mid (pr', r) \in \mathbf{prod}^{\mathcal{P}}(P) \wedge \{\mathbf{A}\} \cup pr' = pr \wr
\end{aligned}$$

□

Lema 5.5.4 Dado el término $P \in \mathbf{SPLA}^{\mathcal{P}}$, la característica $\mathbf{A} \in \mathcal{F}$ y la probabilidad $q \in (0, 1)$, entonces $(pr, p) \in \mathbf{prod}^{\mathcal{P}}(\bar{\mathbf{A}};_q P)$ si y solo si

$$(pr, p) = (\emptyset, 1 - q) \text{ o } p = q \cdot \sum \wr r \mid (pr', r) \in \mathbf{prod}^{\mathcal{P}}(P) \wedge pr' \cup \{\mathbf{A}\} = pr \wr$$

Demostración: Existen dos transiciones a $\bar{\mathbf{A}};_q P$, $\bar{\mathbf{A}};_q P \xrightarrow{\mathbf{A}}_q P$ y $\bar{\mathbf{A}};_q P \xrightarrow{\checkmark}_{1-q} \mathbf{nil}$. Si $\bar{\mathbf{A}};_q P \xRightarrow{s}_r Q$ entonces

1. $s = \checkmark$ y $r = 1 - q$, o
2. $s = \mathbf{A} \cdot s'$, $P \xRightarrow{s}_r Q$, y $r = q \cdot r'$.

Si $pr = \lfloor \mathbf{A} \cdot s' \rfloor$, entonces $pr \neq \emptyset$. Consecuentemente, $(\emptyset, 1 - q) \in \mathbf{prod}^{\mathcal{P}}(\bar{\mathbf{A}};_q P)$, por lo que $pr \neq \emptyset$, entonces $(pr, p) \in \mathbf{prod}^{\mathcal{P}}(\bar{\mathbf{A}};_q P)$, si y solo si

$$\begin{aligned}
d &= \sum \wr r \mid \bar{\mathbf{A}};_q P \xRightarrow{s\checkmark} \mathbf{nil} \wedge \lfloor s \rfloor = pr \wr = \\
&\sum \wr r \mid \bar{\mathbf{A}};_q P \xrightarrow{\mathbf{A}}_q P \xRightarrow{s'\checkmark}_{r'} \mathbf{nil} \wedge \lfloor \mathbf{A} \cdot s' \rfloor = pr \wedge r = q \cdot r' \wr = \\
&\quad \sum \wr r \mid P \xRightarrow{s'\checkmark}_{r'} \mathbf{nil} \wedge \{\mathbf{A}\} \cup \lfloor s' \rfloor = pr \wedge r = q \cdot r' \wr = \\
&\quad \sum \wr r \mid (pr', r') \in \mathbf{prod}^{\mathcal{P}}(P) \wedge \{\mathbf{A}\} \cup pr' = pr \wedge r = q \cdot r' \wr = \\
&\quad q \cdot \sum \wr r' \mid (pr', r') \in \mathbf{prod}^{\mathcal{P}}(P) \wedge \{\mathbf{A}\} \cup pr' = pr \wr
\end{aligned}$$

□

Lema 5.5.5 Dados los términos $P, Q \in \mathbf{SPLA}^{\mathcal{P}}$ y la probabilidad $q \in (0, 1)$, entonces $P \vee_q Q \xRightarrow{s}_r R$, si y solo si

1. $P \xRightarrow{s}_{r'} R$ y $r = q \cdot r'$, o
2. $Q \xRightarrow{s}_{r'} R$ y $r = (1 - q) \cdot r'$

Demostración: Este lema es una consecuencia de las reglas [cho1] y [cho2] de la semántica operacional. \square

Lema 5.5.6 Dados los términos $P, Q \in \text{SPLA}^P$ y la probabilidad $q \in (0, 1)$, entonces $(pr, p) \in \text{prod}^P(P \vee_q Q)$, si y solo si

$$p = \left(q \cdot \sum \{r \mid (pr, r) \in \text{prod}^P(P)\} \right) + \left((1 - q) \cdot \sum \{r \mid (pr, r) \in \text{prod}^P(Q)\} \right)$$

Demostración: $(pr, p) \in \text{prod}^P(P \vee_q Q)$ si y solo si

$$\begin{aligned} p &= \sum \{r \mid P \vee_q Q \xRightarrow{s\checkmark}_r \text{nil}\} = \\ &= \sum \{r \mid (P \xRightarrow{s\checkmark}_{r'} \text{nil} \wedge r = q \cdot r') \vee (Q \xRightarrow{s\checkmark}_{r'} \text{nil} \wedge r = (1 - q) \cdot r')\} = \\ &= \sum \{r \mid P \xRightarrow{s\checkmark}_{r'} \text{nil} \wedge r = q \cdot r'\} + \sum \{r \mid Q \xRightarrow{s\checkmark}_{r'} \text{nil} \wedge r = (1 - q) \cdot r'\} = \\ &= q \cdot \sum \{r \mid P \xRightarrow{s\checkmark}_r \text{nil}\} + (1 - q) \cdot \sum \{r \mid Q \xRightarrow{s\checkmark}_r \text{nil}\} = \\ &= q \cdot \sum \{r \mid (pr, r) \in \text{prod}^P(P)\} + (1 - q) \cdot \sum \{r \mid (pr, r) \in \text{prod}^P(Q)\} \end{aligned}$$

\square

Definición 5.5.7 Dadas las trazas $s, s' \in \mathcal{F}^*$, se define el conjunto de trazas obtenidas de alternar s y s' como $\text{int}(s, s')$. \square

Lema 5.5.8 Dados los términos $P, Q \in \text{SPLA}^P$, las probabilidades $p, q \in (0, 1)$ y las trazas $s, s' \in \mathcal{F}^*$, tal que $p = \sum \{p' \mid P \xRightarrow{s\checkmark}_{p'} \text{nil}\}$ y $q = \sum \{q' \mid Q \xRightarrow{s'\checkmark}_{q'} \text{nil}\}$, entonces

$$p \cdot q = \sum \{r \mid P \wedge Q \xRightarrow{s''\checkmark}_r \text{nil} \wedge s'' \in \text{int}(s, s')\}$$

Demostración: Por inducción de $|s| + |s'|$.

- $|s| + |s'| = 0$: Teniendo en cuenta que $P \wedge Q \xrightarrow{\checkmark}_r \text{nil}$, si y solo si $P \xrightarrow{\checkmark}_{r_1} \text{nil}$, $Q \xrightarrow{\checkmark}_{r_2} \text{nil}$, y $r = r_1 \cdot r_2$, se sostiene la siguiente ecuación:

$$\begin{aligned}
& \sum \wr r \mid P \wedge Q \xrightarrow{\checkmark}_r \text{nil} \wr = \\
& \sum \wr r \mid P \xrightarrow{\checkmark}_{r_1} \text{nil} \wedge Q \xrightarrow{\checkmark}_{r_2} \text{nil} \wedge r = r_1 \cdot r_2 \wr = \\
& \sum \wr r_1 \cdot r_2 \mid P \xrightarrow{\checkmark}_{r_1} \text{nil} \wedge Q \xrightarrow{\checkmark}_{r_2} \text{nil} \wr = \quad (5.1) \\
& \sum \wr r_1 \cdot \left(\sum \wr r_2 \mid Q \xrightarrow{\checkmark}_{r_2} \text{nil} \wr \right) \mid P \xrightarrow{\checkmark}_{r_1} \text{nil} \int = \\
& \sum \wr r_1 \cdot q \mid P \xrightarrow{\checkmark}_{r_1} \text{nil} \wr = q \cdot \sum \wr r_1 \mid P \xrightarrow{\checkmark}_{r_1} \text{nil} \wr = q \cdot p
\end{aligned}$$

- $|s| + |s'| > 0$: Supongamos que $|s| > 0$, en el caso de que $|s'| > 0$ es simétrica. Teniendo que $s = A \cdot s_1$, el multiconjunto $\mathcal{P} = \wr P' \mid P \xrightarrow{A}_{r'} P' \xrightarrow{s_1 \checkmark}_{p'} \text{nil} \wr$, $r_1 = \sum \wr r \mid P \xrightarrow{A}_r P' \wedge P' \in \mathcal{P} \wr$ y $p_1 = \sum \wr r \mid P' \xrightarrow{s_1 \checkmark}_r \text{nil} \wedge P' \in \mathcal{P} \wr$. Es fácil verificar que $p = r_1 \cdot p_1$:

$$\begin{aligned}
p &= \sum \wr r \mid P \xrightarrow{s \checkmark}_r \text{nil} \wr = \\
& \sum \wr r' \cdot p' \mid P \xrightarrow{A}_{r'} P' \xrightarrow{s_1 \checkmark}_{p'} \text{nil} \wr = \\
& \sum \wr r' \cdot \underbrace{\left(\sum \wr p' \mid P' \xrightarrow{s_1 \checkmark}_{p'} \text{nil} \wr \right)}_{p_1} \mid P \xrightarrow{A}_{r'} P' \wedge P' \in \mathcal{P} \int = \quad (5.2) \\
& p_1 \cdot \sum \wr r' \mid P \xrightarrow{A}_{r'} P' \wedge P' \in \mathcal{P} \wr = r_1 \cdot p_1
\end{aligned}$$

Para cualquier $P' \in \mathcal{P}$ por la hipótesis inductiva se obtiene:

$$p' \cdot q = \sum \wr r \mid P' \wedge Q \xrightarrow{s'' \checkmark}_r \text{nil} \wedge s'' \in \text{int}(s_1, s') \wr$$

Entonces,

$$\begin{aligned}
& \sum \wr r \mid P' \in \mathcal{P} \wedge P' \wedge Q \xrightarrow{s'' \checkmark}_r \text{nil} \wedge s'' \in \text{int}(s_1, s') \wr = \\
& \sum \wr p' \cdot q \mid P' \in \mathcal{P} \wedge P' \xrightarrow{s_1 \checkmark}_{p'} \text{nil} \wedge s'' \in \text{int}(s_1, s') \wr \quad (5.3) \\
& q \cdot \sum \wr r \mid P' \xrightarrow{s_1 \checkmark}_{\text{nil}} \text{nil} \wedge P' \in \mathcal{P} \wr = p_1 \cdot q
\end{aligned}$$

Para cualquier $P' \in \mathcal{P}$ se obtiene que $P \wedge Q \xrightarrow{s'' \checkmark}_r \text{nil}$ donde $P \xrightarrow{A}_{r'} P'$, $r = \frac{r'}{2} \cdot p' \cdot q$ y s'' es una alternación de s y s' . Por ello,

$$\begin{aligned}
& \sum \{r \mid P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{r_1}{2}} P' \wedge Q \xRightarrow{s''\checkmark}_{r_2} \mathbf{nil} \wedge P' \xrightarrow{\mathbf{A}}_{r_1} P' \wedge \\
& \quad r = \frac{r_1}{2} \cdot r_1 \wedge s'' \in \text{int}(s_1, s')\} = \\
& \sum \int \left(\sum \int \left[\frac{r'}{2} \cdot r'' \mid P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \wedge Q \xRightarrow{s''\checkmark}_{r''} \mathbf{nil} \wedge s'' \in \text{int}(s_1, s') \right] \right) \int = \\
& \quad \underbrace{\left(\sum \int \left[\frac{r'}{2} \cdot r'' \mid P' \wedge Q \xRightarrow{s''\checkmark}_{r''} \mathbf{nil} \wedge s'' \in \text{int}(s_1, s') \right] \right)}_{\text{hipótesis de inducción: } p_1 \cdot q} \int \left[P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \in \mathcal{P} \right] = \\
& \quad \sum \int \left[\frac{r'}{2} \cdot (p_1 \cdot q) \mid P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \in \mathcal{P} \right] = \\
& \quad \frac{(p_1 \cdot q)}{2} \cdot \sum \{r' \mid P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \in \mathcal{P}\} = \\
& \quad \frac{1}{2}(p_1 \cdot q \cdot r_1) = \frac{1}{2}(p \cdot q) \tag{5.4}
\end{aligned}$$

La ecuación 5.4 agrupa todas las probabilidades de alternar s y s' , donde la primera acción de s se da en primer lugar. Además, es necesario agrupar la información probabilística cuando la primera acción de s' ocurre en primer lugar.

Teniendo en cuenta, si estas acciones existen o no existen dos posibilidades. Dependiendo de s' podemos obtener las siguientes opciones:

- $|s'| = 0$: En este caso tenemos $q = \sum \{q' \mid Q \xrightarrow{\checkmark}_{q'} \mathbf{nil}\}$, Si $Q \xrightarrow{\checkmark}_{q'} \mathbf{nil}$, al aplicar la regla **[con4]** y luego obtener $P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{r' \cdot q'}{2}} P_1$. En este caso se obtiene,

$$\begin{aligned}
& \sum \{r \mid P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{q' \cdot r'}{2}} P' \xRightarrow{s_1\checkmark}_{p'} \mathbf{nil} \wedge r = p' \cdot \frac{q' \cdot r'}{2}\} = \\
& \quad \sum \left[\frac{1}{2} \cdot (p' \cdot r' \cdot q' \mid P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{q' \cdot r'}{2}} P' \xRightarrow{s_1\checkmark}_{p'} \mathbf{nil}) \right] = \\
& \quad \sum \left[\frac{1}{2} \cdot (p' \cdot r' \cdot \underbrace{(\sum \{q' \mid Q \xrightarrow{\checkmark}_{q'} \mathbf{nil}\})}_q \mid P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \xRightarrow{s_1\checkmark}_{p'} \mathbf{nil}) \right] = \\
& \quad \frac{q}{2} \cdot \sum \left[r' \cdot \underbrace{(\sum \{p' \mid P' \xRightarrow{s_1\checkmark}_{p'} \mathbf{nil}\})}_{p_1} \mid P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \in \mathcal{P} \right] = \\
& \quad \frac{1}{2}(q \cdot p_1) \cdot \underbrace{(\sum \{r' \mid P \xrightarrow{\mathbf{A}}_{r'} P' \wedge P' \in \mathcal{P}\})}_{r_1} = \frac{1}{2} \cdot (q \cdot p_1 \cdot r_1) = \frac{1}{2}(q \cdot p) \tag{5.5}
\end{aligned}$$

La única transición de $P \wedge Q$ que permite alternar s y s' tiene la forma

1. $P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{r'}{2}} P' \wedge Q \xRightarrow{s''\checkmark}_{p'} \mathbf{nil}$ como es indicado arriba , o
2. $P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{q' \cdot r'}{2}} P' \xRightarrow{s''\checkmark}_{p'} \mathbf{nil}$.

Entonces,

$$\begin{aligned}
& \sum \{r \mid P \wedge Q \xRightarrow{s\checkmark} \mathbf{nil}\} = \\
& \sum \{r \mid P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{r_1}{2}} P' \wedge Q \xRightarrow{s''\checkmark}_{r_2} \mathbf{nil} \wedge P' \xrightarrow{\mathbf{A}}_{r_1} P' \wedge \\
& \quad r = \frac{r_1}{2} \cdot r_1 \wedge s'' \in \text{int}(s_1, s')\} + \quad (5.6) \\
& \sum \{r \mid P \wedge Q \xrightarrow{\mathbf{A}}_{\frac{q' \cdot r'}{2}} P' \xRightarrow{s_1\checkmark}_{p'} \mathbf{nil} \wedge r = p' \cdot \frac{q' \cdot r'}{2}\} = \\
& \quad \frac{1}{2}(p \cdot q) + \frac{1}{2}(p \cdot q) = p \cdot q
\end{aligned}$$

- $|s'| > 0$: Consideremos $s' = B \cdot s_2$. En este caso, consideramos las alternaciones donde Q produce la característica B en primer lugar. Para ello primero aplicamos la regla **[con2]**. Considerando el multiconjunto $\mathcal{Q} = \{Q' \mid Q \xrightarrow{B}_r Q' \xRightarrow{s_2\checkmark}_q \mathbf{nil}\}$, y dado $r_2 = \sum \{r \mid Q \xrightarrow{B}_r Q' \wedge Q' \in \mathcal{Q}\}$ y $q_1 = \sum \{r \mid Q' \xRightarrow{s_2\checkmark}_r \mathbf{nil} \wedge Q' \in \mathcal{Q}\}$. Con un razonamiento similar a las ecuaciones (5.1) y (5.2) obtenemos:

$$q = r_2 \cdot q_1$$

y

$$\begin{aligned}
& \sum \{r \mid P \wedge Q \xrightarrow{B}_{\frac{r_1}{2}} P \wedge Q' \xRightarrow{s''\checkmark}_{r_2} \mathbf{nil} \wedge \\
& \quad r = \frac{r_1}{2} \cdot r_1 \wedge s'' \in \text{int}(s, s_2)\} = \frac{1}{2}(p \cdot q) \quad (5.7)
\end{aligned}$$

La única transición de $P \wedge Q$ que implica alternar s y s' tiene la forma:

1. $P \wedge Q \xrightarrow{\mathbf{A}}_{p', \frac{r}{2}} P' \wedge Q \xRightarrow{s''\checkmark}_{p'} \mathbf{nil}$ donde s'' es una alternación de s_1 y s' , o
2. $P \wedge Q \xrightarrow{B}_{q', \frac{r}{2}} P \wedge Q' \xRightarrow{s''\checkmark}_{p'} \mathbf{nil}$ donde s'' es una alternación de s y s_2 ,

Entonces,

$$\begin{aligned}
& \sum \{r \mid P \wedge Q \xRightarrow{s\checkmark} \text{nil}\} = \\
& \sum \{r \mid P \wedge Q \xrightarrow{\text{A}}_{\frac{r_1}{2}} P' \wedge Q \xRightarrow{s''\checkmark}_{r_2} \text{nil} \wedge P' \xrightarrow{\text{A}}_{r_1} P' \wedge \\
& \quad r = \frac{r_1}{2} \cdot r_1 \wedge s'' \in \text{int}(s_1, s')\} + \\
& \sum \{r \mid P \wedge Q \xrightarrow{\text{B}}_{\frac{r_1}{2}} P \wedge Q' \xRightarrow{s''\checkmark}_{r_2} \text{nil} \wedge Q' \xrightarrow{\text{B}}_{r_1} Q' \wedge \\
& \quad r = \frac{r_1}{2} \cdot r_1 \wedge s'' \in \text{int}(s, s_2)\} = \\
& \quad \frac{1}{2}(p \cdot q) + \frac{1}{2}(p \cdot q) = p \cdot q
\end{aligned} \tag{5.8}$$

□

Lema 5.5.9 Dado el término $P \in \text{SPLA}^{\mathcal{P}}$, la característica $\text{A} \in \mathcal{F}$ y $P \xRightarrow{s\checkmark}_p \text{nil}$.

1. $\text{A} \in s$, si y solo si $P \Rightarrow \text{A} \xRightarrow{s\checkmark}_p \text{nil}$.
2. $\text{A} \notin s$, si y solo si $P \Rightarrow \text{A} \xRightarrow{s\text{A}\checkmark}_p \text{nil}$.

Demostración: En ambos casos la demostración es simple, por inducción sobre la longitud de $|s|$. □

Lema 5.5.10 Dado el término $P \in \text{SPLA}^{\mathcal{P}}$, la característica $\text{A} \in \mathcal{F}$, la traza $s \in \mathcal{F}^*$ y la probabilidad $p \in (0, 1)$, entonces $P \xRightarrow{s\checkmark}_p \text{nil}$, si y solo si $\text{A} \setminus P \xRightarrow{s\checkmark}_p \text{nil}$ y $\text{A} \notin s$.

Demostración: La demostración es simple, por inducción sobre la longitud de $|s|$. □

Lema 5.5.11 Dado el término $P \in \text{SPLA}^{\mathcal{P}}$, las características $\text{A}, \text{B} \in \mathcal{F}$, la traza $s \in \mathcal{F}^*$ y la probabilidad $p \in (0, 1)$, entonces $P \xRightarrow{s\checkmark}_p \text{nil}$, si y solo si $\text{A} \Rightarrow \text{B}$ in $P \xRightarrow{s'\checkmark}_p \text{nil}$ y s' y tiene una de las siguientes formas: $\text{A} \notin s$ y $s' = s$, $\text{B} \in s$ y $s' = s$ o $\text{A} \in s$, $\text{B} \notin s$ y $s' = s \cdot \text{B}$.

Demostración: La demostración se realiza por inducción sobre la longitud de s .

- $|s| = 0$: En este caso $P \xrightarrow{\checkmark}_p \text{nil}$. Se obtiene el resultado al aplicar la regla [req3].
- $|s| > 0$: Se pueden distinguir tres casos, dependiendo de la primera característica de s :

- $s = As_1$: En este caso existe $p_1, q \in (0, 1)$, tal que $P \xrightarrow{A}_{p_1} P_1 \xRightarrow{s_1\checkmark}_q \text{nil}$. Cuando se aplica la regla [req2] se obtiene $A \Rightarrow B \text{ in } P \xrightarrow{A}_{p_1} P_1 \Rightarrow B$. Se obtiene el resultado al aplicar el lema 5.5.9.
- $s = Bs_1$: En este caso existe $p_1, q \in (0, 1)$, tal que $P \xrightarrow{A}_{p_1} P_1 \xRightarrow{s_1\checkmark}_q \text{nil}$. Al aplicar la regla [req2] se obtiene $A \Rightarrow B \text{ in } P \xrightarrow{B}_{p_1} P_1 \Rightarrow A$. Se obtiene el resultado al aplicar el lema 5.5.9.
- $s = Cs_1$ con $C \neq A$ y $C \neq B$: En este caso existe $p_1, q \in (0, 1)$, tal que $P \xrightarrow{C}_{p_1} P_1 \xRightarrow{s_1\checkmark}_q \text{nil}$. Al aplicar la regla [req1], se obtiene $A \Rightarrow B \text{ in } P \xrightarrow{C}_{p_1} A \Rightarrow B \text{ in } P_1$, obteniendo el resultado de aplicar la hipótesis inductiva sobre s_1 .

□

Lema 5.5.12 Dado el término $P \in \text{SPLA}^P$, las características $A, B \in \mathcal{F}$, $s \in \mathcal{F}^*$ y la probabilidad $p \in (0, 1)$, entonces $P \xRightarrow{s\checkmark}_p \text{nil}$, si y solo si $A \not\Rightarrow B \text{ in } P \xRightarrow{s\checkmark}_p \text{nil}$, $A \notin s$ y $B \notin s$.

Demostración: La demostración se realiza por inducción sobre la longitud de s .

- $|s| = 0$: En este caso $P \xrightarrow{\checkmark}_p \text{nil}$. Se obtiene el resultado de aplicar la regla [excl4].
- $|s| > 0$: Se distinguen los siguientes casos dependiendo de la primera característica de s :
 - $s = As_1$: En este caso existe $p_1, q \in (0, 1)$ tal que $P \xrightarrow{A}_{p_1} P_1 \xRightarrow{s_1\checkmark}_q \text{nil}$. Al aplicar la regla [req2] se obtiene $A \Rightarrow B \text{ in } P \xrightarrow{A}_{p_1} P_1 \setminus B$. Según el lema 5.5.9,
 - $B \in s_1$, si y solo si $P_1 \Rightarrow B \xRightarrow{s_1\checkmark}_q \text{nil}$.
 - $B \notin s_1$, si y solo si $P_1 \Rightarrow B \xRightarrow{s_1B\checkmark}_q \text{nil}$.
 - $s = Cs_1$ con $C \neq A$: En este caso es $p_1, q \in (0, 1)$ tal que $P \xrightarrow{C}_{p_1} P_1 \xRightarrow{s_1\checkmark}_q \text{nil}$. Al aplicar la regla [req1], se obtiene $A \Rightarrow B \text{ in } P \xrightarrow{C}_{p_1} A \Rightarrow B \text{ in } P_1$, y entonces el resultado es obtenido al aplicar la hipótesis inductiva sobre s_1 .

□

5.6. Ocultando conjuntos de características

El cálculo de la probabilidad de una característica dentro de una SPL permite determinar cuan utilizado será un componente dentro de la familia de productos. Por ejemplo, en el caso del proceso de *testing*, es interesante conocer las características más frecuentes para centrar el análisis en estos componentes. Esta probabilidad se ve afectada por los operadores sintácticos que influyen en la cardinalidad del número de productos válidos a generar. Esta condición es de suma importancia para la optimización de los términos del álgebra, ya que de esa manera todo elemento sintáctico que no influya en la cardinalidad del conjunto de productos válidos, puede ser eliminado del modelo. Para calcular la probabilidad de un conjunto de características se procederá a *ocultar* otras. Debido a que, por razones de coste computacional no es posible procesar todos los productos de la SPL. Sin embargo, esperamos aliviar este inconveniente si restringimos el análisis a un subconjunto de características. Así, todas las demás características serán transformadas en una nueva, mostrada como $\perp \notin \mathcal{F}$ y considerando el conjunto $\mathcal{F}_\perp = \mathcal{F} \cup \{\perp\}$.

El conjunto de operadores se ha extendido añadiendo uno nuevo. Este nuevo operador permite ocultar un conjunto de características dentro de un término.

$$\begin{array}{ll}
 \text{[hid1]} & \frac{P \xrightarrow{\mathbf{A}}_p P', \mathbf{A} \in \mathcal{A}}{P[\mathcal{A}] \xrightarrow{\perp}_p P'[\mathcal{A}]} \\
 \text{[hid2]} & \frac{P \xrightarrow{\mathbf{A}}_p P', \mathbf{A} \notin \mathcal{A}}{P[\mathcal{A}] \xrightarrow{\mathbf{A}}_p P'[\mathcal{A}]}
 \end{array}$$

Figura 5.5: Semántica operacional para el operador de ocultamiento.

Definición 5.6.1 Dado un subconjunto de características $\mathcal{A} \subseteq \mathcal{F}$ y un término $P \in \text{SPLA}^{\mathcal{P}}$, entonces $P[\mathcal{A}]$ describe el ocultamiento de las características en \mathcal{A} para el término P . \square

Es necesario definir la semántica del nuevo operador. Primero, la semántica operacional se obtiene de las reglas descritas en la figura 5.5. Para poder definir la semántica denotacional

del nuevo operador, es necesario definir una función auxiliar que oculte ciertas características dado un producto.

Definición 5.6.2 Dado el producto $pr \subseteq \mathcal{F}$ y un conjunto de características $\mathcal{A} \subseteq \mathcal{F}$, el *ocultamiento del conjunto \mathcal{A} en pr* , escrito como $pr[\mathcal{A}]$, se define de la siguiente manera:

$$pr[\mathcal{A}] = \{\mathbf{A} \mid \mathbf{A} \in pr \wedge \mathbf{A} \notin \mathcal{A}\} \cup \begin{cases} \{\perp\} & \text{si } pr \cap \mathcal{A} \neq \emptyset \\ \emptyset & \text{si } pr \cap \mathcal{A} = \emptyset \end{cases}$$

De manera análoga, para cualquier secuencia $s \in \mathcal{F}^*$, se considera que $s[\mathcal{A}]$ describe la traza producida a partir de s una vez reemplazadas todas las ocurrencias de las características pertenecientes a \mathcal{A} por el símbolo \perp en s . \square

Definición 5.6.3 Dado el multiconjunto $M \in \mathcal{M}$ y un conjunto de características $\mathcal{A} \subseteq \mathcal{F}$, se define:

$$\llbracket \cdot[\mathcal{A}] \rrbracket^{\mathcal{P}} = \text{accum}\left(\lambda(pr, p) \mid (pr, p) \in M\right)$$

\square

Finalmente, es necesario demostrar que la semántica operacional y la semántica denotacional coinciden. La demostración del siguiente resultado es una consecuencia inmediata de la proposición 5.6.5.

Proposición 5.6.4 Dado el subconjunto de características $\mathcal{A} \subseteq \mathcal{F}$ y el término $P \in \text{SPLA}^{\mathcal{P}}$, se obtiene que $\text{prod}^{\mathcal{P}}(P[\mathcal{A}]) = \llbracket \text{prod}^{\mathcal{P}}(P)[\mathcal{A}] \rrbracket^{\mathcal{P}}$.

Demostración: $(pr, p) \in \text{prod}^{\mathcal{P}}(P[\mathcal{A}])$, si y solo si

$$\begin{aligned} p &= \sum \lambda r \mid P[\mathcal{A}] \xRightarrow{s\checkmark}_r P'[\mathcal{A}]. \text{ } pr = \lfloor s \rfloor \rfloor = \\ &\sum \lambda r \mid P \xRightarrow{s'\checkmark}_r P', \text{ } s = s'[\mathcal{A}], \text{ } pr = \lfloor s \rfloor \rfloor = \\ &\sum \lambda r \mid P \xRightarrow{s'\checkmark}_r P', \text{ } s = pr[\mathcal{A}] \rfloor = \\ &\sum \lambda r \mid (pr', r) \in \text{prod}^{\mathcal{P}}(P), \text{ } pr' = pr[\mathcal{A}] \rfloor = \end{aligned}$$

Entonces, $(pr, p) \in \text{prod}^{\mathcal{P}}(P[\mathcal{A}])$, si y solo si $(pr, p) \in \llbracket (\text{prod}^{\mathcal{P}}(P))[\mathcal{A}] \rrbracket^{\mathcal{P}}$

\square

Proposición 5.6.5 $P[\mathcal{A}] \xRightarrow{s}_r Q[\mathcal{A}]$, si y solo si $r = \sum \{p \mid P \xRightarrow{s'}_p Q, s = s'[\mathcal{A}]\}$

Demostración: La demostración se obtiene por inducción sobre la longitud de la traza s . Si la longitud es cero, es trivial. Se supone que $s = A \cdot s_1$. Si $A = \perp$, entonces cualquier transición $P[\mathcal{A}] \xRightarrow{s}_p Q[\mathcal{A}]$ puede ser dividida en transiciones, posiblemente más de una. Por ejemplo.

$$P[\mathcal{A}] \xrightarrow{\perp}_{r_1} P_1[\mathcal{A}] \xRightarrow{s_1}_{r_2} Q$$

de lo cual obtenemos

$$\begin{aligned} r = \sum \{p \mid P[\mathcal{A}] \xRightarrow{s}_p Q\} &= \sum \{r_1 \cdot r_2 \mid P[\mathcal{A}] \xrightarrow{\perp}_{r_1} P_1[\mathcal{A}] \xRightarrow{s_1}_{r_2} Q\} = \\ &= \sum \{r'_1 \cdot r_2 \mid P[\mathcal{A}] \xrightarrow{B}_{r'_1} P'_1[\mathcal{A}] \xRightarrow{s_1}_{r_2} Q, B \in \mathcal{A}\} \end{aligned}$$

Para cada una de las r'_1 , es posible aplicar el método inductivo a cada una de las transiciones $P'_1[\mathcal{A}] \xRightarrow{s_1}_{r_2} Q$ para obtener $r_2 = \sum \{r_2' \mid P_1 \xRightarrow{s'_1}_{r_2'} Q, s_1 = s'_1[\mathcal{A}]\}$. Continuando la última ecuación:

$$\begin{aligned} &\sum \{r'_1 \cdot r_2 \mid P[\mathcal{A}] \xrightarrow{B}_{r'_1} P'_1[\mathcal{A}] \xRightarrow{s_1}_{r_2} Q, B \in \mathcal{A}\} = \\ &\sum \{r'_1 \cdot r_2' \mid P[\mathcal{A}] \xrightarrow{B}_{r'_1} P'_1 \xRightarrow{s'_1}_{r_2'} Q, B \in \mathcal{A}, s_1 = s'_1[\mathcal{A}]\} = \\ &\sum \{r_1 \cdot r_2' \mid P[\mathcal{A}] \xrightarrow{\perp}_{r_1} P_1 \xRightarrow{s'_1}_{r_2'} Q, B \in \mathcal{A}, s_1 = s'_1[\mathcal{A}]\} = \\ &\sum \{r \mid P \xRightarrow{s'}_r Q, s = s'[\mathcal{A}]\} \end{aligned}$$

El caso cuando $A \notin \mathcal{A}$ es similar al anterior, solo es necesario saltar el paso de B a \perp . \square

Como es usual en las álgebras de procesos, es deseable que el operador de ocultamiento sea *derivado*, es decir, que dado un término sintáctico, exista un término semánticamente equivalente sin la ocurrencia del operador de ocultamiento. La siguiente proposición muestra que este es el caso. Es posible *eliminar* el operador de ocultamiento de cualquier término. La idea es sustituir cualquier ocurrencia de las acciones ocultas por el símbolo \perp , teniendo en cuenta que no es posible ocultar acciones que aparezcan en los operadores de restricción. La demostración del siguiente resultado es simple por inducción estructural y utilizando la proposición 5.6.5.

Teorema 5.6.6 Dados los términos $P, Q \in \text{SPLA}^P$, la probabilidad $r \in (0, 1]$ y el conjunto de acciones ocultas $\mathcal{A} \subseteq \mathcal{F}$, se obtienen los siguientes resultados:

$$\begin{aligned}
\llbracket \checkmark[\mathcal{A}] \rrbracket^P &= \llbracket \checkmark \rrbracket^P \\
\llbracket \text{nil}[\mathcal{A}] \rrbracket^P &= \llbracket \text{nil} \rrbracket^P \\
\llbracket (A; P)[\mathcal{A}] \rrbracket^P &= \begin{cases} \llbracket \perp; (P[\mathcal{A}]) \rrbracket^P & \text{si } A \in \mathcal{A} \\ \llbracket A; (P[\mathcal{A}]) \rrbracket^P & \text{si } A \notin \mathcal{A} \end{cases} \\
\llbracket (\bar{A};_r P)[\mathcal{A}] \rrbracket^P &= \begin{cases} \llbracket \bar{\perp};_r (P[\mathcal{A}]) \rrbracket^P & \text{si } A \in \mathcal{A} \\ \llbracket \bar{A};_r (P[\mathcal{A}]) \rrbracket^P & \text{si } A \notin \mathcal{A} \end{cases} \\
\llbracket (P \vee_P Q)[\mathcal{A}] \rrbracket^P &= \llbracket (P[\mathcal{A}]) \vee_P (Q[\mathcal{A}]) \rrbracket^P \\
\llbracket (P \wedge Q)[\mathcal{A}] \rrbracket^P &= \llbracket (P[\mathcal{A}]) \wedge (Q[\mathcal{A}]) \rrbracket^P \\
\text{Si } A, B \notin \mathcal{A} \text{ entonces } \llbracket (A \Rightarrow B \text{ in } P)[\mathcal{A}] \rrbracket^P &= \llbracket A \Rightarrow B \text{ in } (P[\mathcal{A}]) \rrbracket^P \\
\text{Si } A, B \notin \mathcal{A} \text{ entonces } \llbracket (B \not\Rightarrow P \text{ in })[\mathcal{A}] \rrbracket^P &= \llbracket A \not\Rightarrow B \text{ in } (P[\mathcal{A}]) \rrbracket^P
\end{aligned}$$

□

Este último teorema nos permite hacer una implementación eficiente del cálculo de la probabilidad de una determinada característica. Esta probabilidad se calcula mediante el ocultamiento del resto de características. El problema reside en que no es posible ocultar todas las características. Tal y como muestran las condiciones del teorema anterior, vemos que en el caso de los requisitos y las prohibiciones, no se pueden ocultar todas las características. Por ello, deben estudiarse las relaciones de aquellas características que quieran, o no, ocultarse.

5.7. Implementación

Esta sección presenta los resultados obtenidos de la implementación de la semántica denotacional para la extensión probabilística². El software está licenciado bajo GPL v3 (mas detalles en <http://www.gnu.org/copyleft/gpl.html>). La herramienta contiene el código fuente de la implementación, las instrucciones sobre su uso y archivos de ejemplo

²El código fuente de la implementación puede ser descargado directamente desde la siguiente URL: <http://ccamacho.github.io/phd/splp/>.

para reproducir los resultados de las simulaciones. De forma similar a como se hizo en secciones anteriores, en esta sección se presentan detalles sobre posibles usos prácticos. Para ello, se ha utilizado un modelo de variabilidad con 1500 características. El modelo ha sido creado utilizando el el generador de modelos de características BeTTy [96]. Los parámetros de configuración utilizados en BeTTy han sido los siguientes:

- La probabilidad de que exista una característica obligatoria es de 0.2.
- La probabilidad de que exista una característica opcional es de 0.3.
- La probabilidad de que una característica esté dentro de una relación de *selección única* es de 0.25.
- La probabilidad de que una característica esté dentro de una relación de *paralelo* es de 0.25.

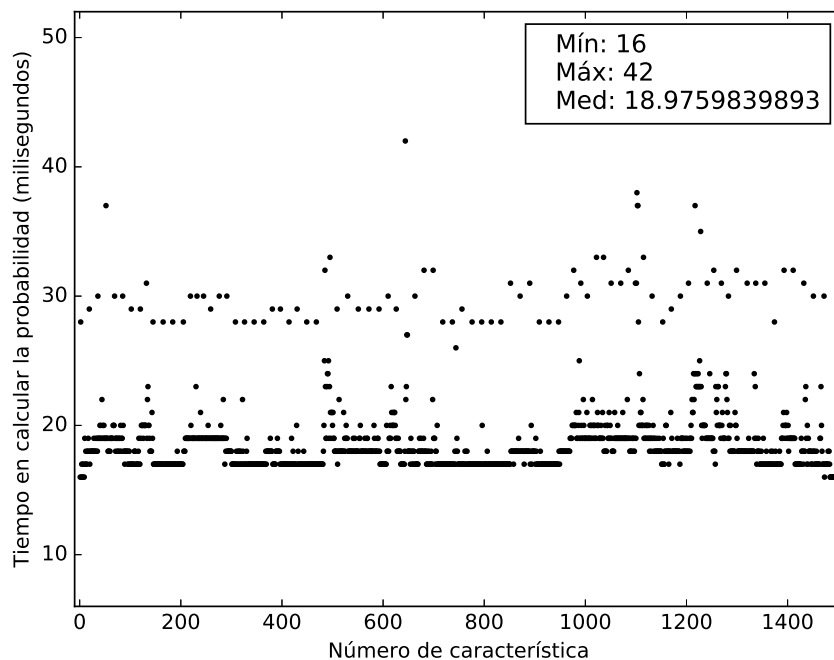


Figura 5.6: Tiempo de procesamiento para un modelo con 1500 características.

Existen dos aplicaciones prácticas dentro del análisis de líneas de productos usando métodos probabilísticos. La primera consiste en calcular la probabilidad de una característica. Esto permitirá asignar recursos de manera más eficiente al probar las características con mayor probabilidad de ocurrencia dentro de la línea de productos. Otra aplicación consiste en estimar la cobertura de las pruebas dentro de las líneas de productos, ya que es posible determinar el número de productos que serán abarcados por las pruebas.

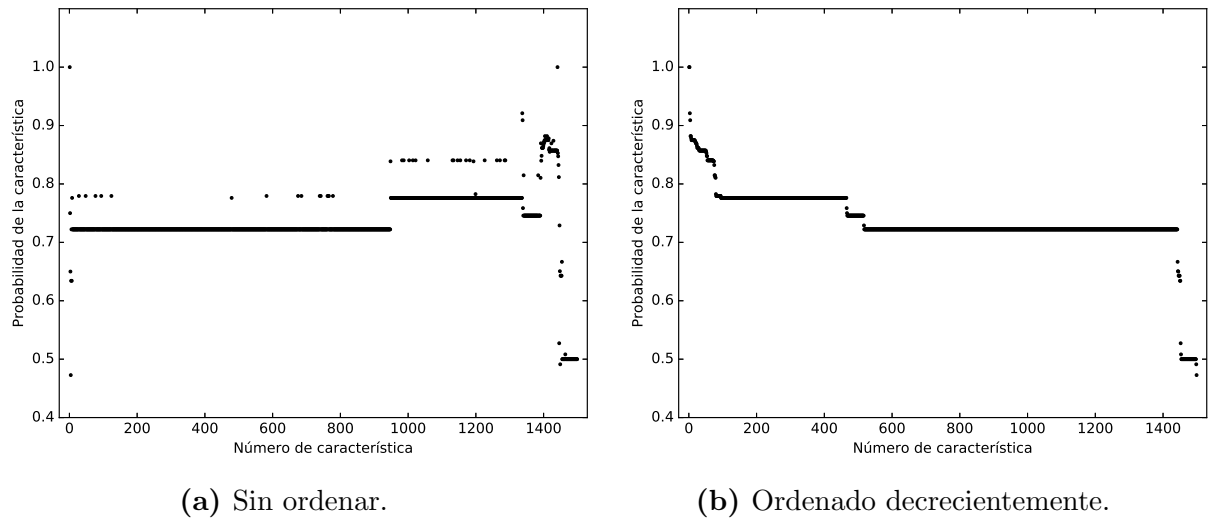


Figura 5.7: Probabilidad de las características del modelo de variabilidad.

La figura 5.6 muestra el tiempo en calcular la probabilidad de cada característica dentro del modelo de características. En ésta gráfica se puede apreciar que existe una variación entre el tiempo en calcular la probabilidad de cada característica. Esto se debe al estado actual del nodo donde se ha ejecutado la simulación. Puesto que los tiempos de ejecución son pequeños cualquier retraso en la planificación del proceso puede tener como consecuencia una latencia en su ejecución, lo que repercute en el resultado final. Esta variación puede considerarse insignificante, ya que los tiempos de ejecución están en el rango de 15 y 38 milisegundos para cada característica dentro del modelo.

El modelo utilizado en la simulación ha sido ejecutado en 10 ocasiones. En todas las ejecuciones se obtienen resultados similares en la banda de 15 y 20 milisegundos, con algunas características procesándose en la banda superior entre 20 y 38 milisegundos.

La figura 5.7a muestra, gráficamente, la probabilidad de cada característica dentro del modelo a medida que éste se procesa. Gracias a esta representación gráfica es posible distinguir claramente agrupaciones de características con mayor probabilidad. La figura 5.7b muestra los resultados de la figura 5.7a ordenados de manera descendiente. Esto permite apreciar con mayor claridad las características con mayor probabilidad de ocurrencia.

En este caso particular, es posible identificar que las características con probabilidad mayor a 0.75 son sólo 450 características de las 1500 características existentes en el modelo. Este análisis permitiría establecer que al realizar pruebas unitarias sobre ese 28 % de las características con probabilidad mayor a 0.75, se incluirían pruebas que abarcarían los componentes de al menos el 75 % de los productos generados.

Capítulo 6

Conclusiones

“Where you start is not as important as where you finish.”

Zig Ziglar

Contenido

6.1. Contribuciones principales	154
6.1.1. SPLA: Marco teórico para representar diagramas FODA	154
6.1.2. SPLA ^C : Extensión de SPLA para representar costes	155
6.1.3. SPLA ^P : Extensión de SPLA para representar probabilidades	156
6.2. Publicaciones obtenidas con esta tesis doctoral	157
6.3. Trabajo futuro	158

A lo largo de los últimos 30 años han sido propuestos una amplia variedad de modelos para representar SPLs. Sin embargo, el análisis formal y automático de los modelos de variabilidad sigue siendo un desafío donde, dependiendo de la información que sea necesario generar, surgen inconvenientes ligados a la explosión combinatoria ocasionada por ciertos operadores del álgebra. Este trabajo propone y muestra un formalismo matemático capaz de definir modelos de variabilidad. El formalismo propuesto es lo suficientemente flexible como para poder ser aplicado a otros ámbitos fuera del desarrollo de *software*, ya que su aplicación no se basa exclusivamente en las reglas definidas en FODA. Además, se muestran distintas extensiones e implementaciones que permiten generar información útil para la toma de decisiones.

6.1. Contribuciones principales

En este trabajo de investigación se ha definido un marco teórico para modelar líneas de productos software, utilizando las álgebras de procesos como base teórica y FODA como modelo de variabilidad de referencia.

6.1.1. SPLA: Marco teórico para representar diagramas FODA

En la primera fase de esta tesis doctoral se desarrolló un marco teórico para modelar y representar formalmente diagramas FODA [4]. Este formalismo comienza con la definición de la sintaxis del lenguaje. Una vez definidos los operadores sintácticos, se procedió a describir el proceso de traducción de FODA a SPLA. Seguidamente se presentó un mecanismo que permitiera comprobar factibilidad de un término del álgebra, esto es, comprobar si un término sintácticamente correcto es capaz de generar productos válidos. Posteriormente se definieron tres semánticas dentro del marco formal, la semántica operacional, la semántica denotacional y la semántica axiomática. En la semántica operacional se ha descrito el mecanismo para calcular todos los productos válidos de la SPL, teniendo en cuenta el orden en el que se procesan las características. En la semántica denotacional los productos han sido contruidos como conjuntos de características y, por lo tanto, no se ha tenido en cuenta el orden en el que éstos se producen. La semántica axiomática ha sido definida para describir los axiomas necesarios para garantizar que los planteamientos descritos son completos y correctos.

Una vez presentada la descripción de las semánticas y demostrado que la traducción de un diagrama FODA es consistente al traducirse a SPLA, se demostró que a partir de un diagrama FODA pueden generarse distintos términos SPLA equivalentes entre sí. Además, al existir distintas representaciones de un mismo término, se demostró que éstas pueden optimizarse o reducirse, siendo equivalentes a los originales tomados del diagrama FODA. Para ello, se utilizan únicamente los elementos básicos del álgebra. Una vez terminada la especificación formal del método, se procedió a describir un caso de estudio utilizando

un sistema de *streaming* de video. El modelo de variabilidad de este sistema fue definido utilizando FODA. Seguidamente, se desarrolló el modelo equivalente en SPLA, así como el mecanismo de procesamiento basado en las reglas de la semántica operacional. Una vez desarrollado el modelo, se procedió a describir la implementación del formalismo, el cual está formado por dos módulos. El primer módulo permite comprobar la factibilidad de un término del álgebra, esto es, comprobar si éste genera productos válidos y el segundo módulo describe la ejecución de las reglas de la semántica denotacional.

Los resultados obtenidos tras el desarrollo de los conceptos detallados en este apartado han sido publicados en ① (ver sección 6.2).

6.1.2. SPLA^C: Extensión de SPLA para representar costes

Una vez definido SPLA, se procedió a definir la extensión que permite representar el coste de procesar una característica en un producto en construcción [30]. Debido a la necesidad de comparar distintos productos válidos en función de su coste de producción, se ha representado el coste dentro de las reglas semánticas del álgebra. Para ello, con el objetivo de tener en cuenta el coste, se han actualizado las reglas de la semántica operacional y de la semántica denotacional. Una vez actualizadas las reglas semánticas, se ha planteado un caso de estudio modelando un sistema de gestión de la configuración con *Chef.io*. Además, se ha realizado la implementación de la semántica operacional modelando el coste de cada producto válido. Es importante destacar que la semántica operacional presenta un problema combinatorio al procesar los operadores de paralelo y opcional. En este punto surge la necesidad de describir distintos mecanismos que permitan que estos operadores tengan el menor impacto posible al procesar el modelo. Para minimizar el efecto de estos operadores se recomienda optimizar el modelo de características antes de su procesamiento, eliminando o reduciendo los operadores que agrupan términos en paralelo o de manera opcional. Seguidamente, se ha mostrado que es posible procesar los términos del álgebra dentro de un sistema distribuido, distribuyendo la carga entre los distintos nodos del *cluster*, para así in-

crementar el rendimiento del sistema. Finalmente se ha mostrado, en la implementación, la opción de configurar umbrales de coste para procesar los modelos, de tal forma que si se supera el coste estimado, las ramas correspondientes dentro del árbol de trazas no serán procesadas. De igual manera se define un umbral mínimo, donde únicamente los productos que superen ese umbral serán considerados válidos.

Los resultados obtenidos tras el desarrollo de los conceptos explicados en este apartado han sido publicados en ② (ver sección 6.2).

6.1.3. $SPLA^P$: Extensión de SPLA para representar probabilidades

Con el objetivo de aliviar los problemas asociados a la explosión combinatoria de algunos operadores del álgebra, es necesario generar información relevante para determinar tanto la probabilidad de que una característica cualquiera se encuentre presente en el conjunto de los productos válidos del término estudiado, como la probabilidad de generar un producto válido dentro de la SPL [31]. Para ello, se han propuesto nuevos operadores sintácticos en la extensión probabilística. Una vez definidos, se han creado las extensiones para las reglas de las semánticas descritas en secciones anteriores, tales como la semántica operacional y la semántica denotacional. Una vez proporcionada la definición formal de la extensión probabilística, es imprescindible mostrar que la extensión es consistente con las definiciones de las secciones anteriores. Por ello, una vez definida la extensión probabilística, se han descrito las ventajas de ocultar aquellos elementos sintácticos que no afectan la cardinalidad del conjunto de productos válidos del sistema. Para mostrar la aplicabilidad de este método, se ha desarrollado una herramienta con la implementación de la semántica denotacional para la extensión probabilística. El resultado de esta extensión muestra que realizando análisis probabilístico, es posible calcular la probabilidad de aparición de cada característica en los productos válidos del modelo, y así centrar el análisis de las pruebas sobre las características más comunes o probables.

Los resultados obtenidos tras el desarrollo de los conceptos descritos en este apartado se

encuentran disponibles en ③ (ver sección 6.2).

6.2. Publicaciones obtenidas con esta tesis doctoral

Las contribuciones de este trabajo de investigación han sido descritas de acuerdo a como han sido publicadas cronológicamente [4, 30, 31]. La información sobre el índice de impacto y cuartil, ha sido extraída de la página oficial de la Fundación Española para la Ciencia y la Tecnología (FECYT)¹.

- ① C. Andrés, C. Camacho y L. Llana. A formal framework for software product lines. *Information and Software Technology* 55 (11) (2013) 1925–1947. doi:10.1016/j.infsof.2013.05.005
 - *Ranking* JCR 2013: 31/105.
 - Cuartil: Q2.
 - Índice de impacto: 1,328.
 - En: Computer Science, Software Engineering.
- ② C. Camacho, L. Llana y A. Núñez. Cost-related interface for software product lines. *Journal of Logical and Algebraic Methods in Programming* 85 (2016) 227–244. doi:10.1016/j.jlamp.2015.09.009
 - *Ranking* JCR 2015: 5/22.
 - Cuartil: Q1.
 - Índice de impacto: 0,636.
 - En: Logic.
- ③ C. Camacho, L. Llana, A. Núñez y M. Núñez. Probabilistic software product lines

Esta tesis doctoral y todos sus soportes asociados se encuentran disponibles en la siguiente URL: <http://ccamacho.github.io/phd/>.

¹<https://www.recursoscientificos.fecyt.es/servicios/indices-de-impacto>.

6.3. Trabajo futuro

Tras observar el inconveniente de la explosión combinatoria al aplicar algunos operadores, se plantea procesar los modelos de variabilidad reduciendo en la manera de lo posible su coste computacional, intentando aliviar el problema de la explosión combinatoria. Entre las posibles soluciones podrían estudiarse diferentes heurísticas que permitan, por ejemplo, eliminar características no utilizables.

De igual manera, se considera adaptar el formalismo presentado en esta tesis a herramientas de análisis como Uppaal Cora [1], ya que el modelo definido en SPLA es similar al definido en [20].

Además, planeamos desarrollar futuros mecanismos que permitan simplificar y optimizar términos del álgebra, basando su análisis, en métodos probabilísticos.

Posibles extensiones de este trabajo de investigación, consistirán en modelar sistemas de computación en la nube² (del inglés *cloud computing*) utilizando las herramientas y formalismos ya definidos.

También, resulta interesante el análisis en tiempo real de modelos de variabilidad que puedan cambiar de forma dinámica. Por ejemplo, al modelar los componentes en uso de una nube de cómputo³ (del inglés *cloud*). Ya que estos pueden variar de forma dinámica de acuerdo a la carga de trabajo de la misma, y en función de estos cambios, desencadenar procesos de actualización de las estructuras previamente definidas para validarlas [85].

Hasta ahora, el uso de las herramientas desarrolladas ha sido mediante línea de comandos, también se plantea a futuro, desarrollar herramientas con interfaces gráficas para proveer al formalismo de una mejor usabilidad.

De igual manera, para todas las extensiones planteadas SPLA se deben buscar posibles aplicaciones prácticas.

²Este término hace referencia al consumir recursos informáticos como servicios, en vez de ser mantenidos por el usuario en sí.

³Espacio de almacenamiento y procesamiento de datos y archivos ubicado en internet, al que puede acceder el usuario desde cualquier dispositivo.

Bibliografía

- [1] UPPAAL CORA. <http://people.cs.aau.dk/~adavid/cora/index.html> (2005).
- [2] Chef.io. <http://www.chef.io/> (2015).
- [3] R. Alur, S. La Torre y G. Pappas. Optimal paths in weighted timed automata. En: MariaDomenica Di Benedetto y Alberto Sangiovanni-Vincentelli (Eds.), *Hybrid Systems: Computation and Control*. Vol. 2034 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001. pp. 49–62. doi:10.1007/3-540-45351-2_8.
- [4] C. Andrés, C. Camacho y L. Llana. A formal framework for software product lines. *Information and Software Technology* 55 (11) (2013) 1925–1947. doi:10.1016/j.infsof.2013.05.005.
- [5] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. Royer, A. Rummler y A. Sousa. A model-driven traceability framework for software product lines. *Software and Systems Modeling* 9 (4) (2010) 427–451. doi:10.1007/s10270-009-0120-9.
- [6] A. Antonik, M. Huth, K.G. Larsen, U. Nyman y A. Wasowski. Complexity of decision problems for mixed and modal specifications. En: *11th International Conference on Foundations of Science and Computation Structures and Theory and practice of software, FOSSACS’08/ETAPS’08*. Springer, 2008. pp. 112–126. doi:10.1007/978-3-540-78499-9_9.
- [7] H. Arboleda y J. Royer. *Model-Driven and Software Product Line Engineering*. Wiley, 2012. doi:10.1002/9781118561379.fmatter.
- [8] P. Asirelli, M. H. ter Beek, A. Fantechi y S. Gnesi. A logical framework to deal with variability. En: *8th International Conference on Integrated Formal Methods, IFM’10*. Springer, 2010. pp. 43–58. doi:10.1007/978-3-642-16265-7_5.

- [9] P. Asirelli, M. H. ter Beek, A. Fantechi, S. Gnesi y F. Mazzanti. Design and validation of variability in product lines. En: 2nd International Workshop on Product Line Approaches in Software Engineering, PLEASE '11. ACM, 2011. pp. 25–30. doi:10.1145/1985484.1985492.
- [10] P. Asirelli, M. H. ter Beek, S. Gnesi y A. Fantechi. A deontic logical framework for modelling product families. En: 4th International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS'10. 2010. pp. 37–44.
- [11] P. Asirelli, M. H. ter Beek, S. Gnesi y A. Fantechi. Formal description of variability in product families. En: 15th International Software Product Line Conference, SPLC '11. IEEE Computer Society Press, 2011. pp. 130–139. doi:10.1109/SPLC.2011.34.
- [12] C. Atkinson y D. Muthig. Component-based product line engineering with UML. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-46020-9_34.
- [13] J.C.M. Baeten, J.A. Bergstra y S.A. Smolka. Axiomatizing probabilistic processes: Acp with generative probabilities. Information and Computation 121 (2) (1995) 234 – 255. doi:http://dx.doi.org/10.1006/inco.1995.1135.
- [14] L. Bass, P. Clements y R. Kazman. Software Architecture in Practice. 3rd Edition. Addison-Wesley Professional, 2012.
- [15] D. Batory. Feature models, grammars, and propositional formulas. En: 9th International Software Product Line Conference, SPLC'05. Springer, 2005. pp. 7–20. doi:10.1007/11554844_3.
- [16] D. Batory, D. Benavides y A. Ruiz Cortés. Automated analysis of feature models: challenges ahead. Commun. ACM 49 (12) (2006) 45–47. doi:10.1145/1183264.
- [17] D. Batory, D. Benavides y A. Ruiz. Automated analysis of feature models: challenges ahead. Communications of the ACM 49 (2006) 45–47. doi:10.1145/1183236.1183264.

- [18] G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi y M. Hendriks. Uppaal 4.0. En: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems. QEST '06. IEEE Computer Society, Washington, DC, USA, 2006. pp. 125–126. doi:10.1109/QEST.2006.59.
- [19] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn y F. Vaandrager. Minimum-cost reachability for priced time automata. En: MariaDomenica Di Benedetto y Alberto Sangiovanni-Vincentelli (Eds.), Hybrid Systems: Computation and Control. Vol. 2034 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001. pp. 147–161. doi:10.1007/3-540-45351-2_15.
- [20] G. Behrmann, K. G. Larsen y J. I. Rasmussen. Optimal scheduling using priced timed automata. SIGMETRICS Performance Evaluation Review 32 (4) (2005) 34–40. doi:10.1145/1059816.1059823.
- [21] D. Benavides, S. Segura y A. Ruiz. Automated analysis of feature models 20 years later: A literature review. Information Systems 35 (6) (2010) 615–636. doi:10.1016/j.is.2010.01.001.
- [22] A. Bertolino y S. Gnesi. Pluto: A test methodology for product families. En: 5th International Workshop Software Product-Family Engineering, PFE'03. Springer, 2004. pp. 181–197. doi:10.1007/978-3-540-24667-1_14.
- [23] G. Bockle, P. Clements, J. McGregor, D. Muthig y K. Schmid. A cost model for software product lines. En: FrankJ. Linden (Ed.), Software Product-Family Engineering. Vol. 3014 of LNCS. Springer Berlin Heidelberg, 2004. pp. 310–316. doi:10.1007/978-3-540-24667-1_23.
- [24] B. Boehm, C. Abts, A. Brown y S. Chulani. Software Cost Estimation with COCOMO II. 1st Edition. Prentice Hall Press, 2009.
- [25] Y. Bontemps, P. Heymans, P. Schobbens y J.C. Trigaux. Semantics of FODA feature diagrams. En: 1st Workshop on Software Variability Management for Product Derivation – Towards Tool Support, SPLCW'04. Springer, 2004. pp. 48–58.

- [26] J. Bosch. Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
- [27] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis y S. Yovine. Kronos: A model-checking tool for real-time systems. En: AndersP. Ravn y Hans Rischel (Eds.), Formal Techniques in Real-Time and Fault-Tolerant Systems. Vol. 1486 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998. pp. 298–302. doi:10.1007/BFb0055357.
- [28] V. Braberman, A. Olivero y F. Schapachnik. Dealing with practical limitations of distributed timed model checking for timed automata. Formal Methods in System Design 29 (2) (2006) 197–214. doi:10.1007/s10703-006-0012-3.
- [29] P. Buchholz y P. Kemper. Quantifying the dynamic behavior of process algebras. En: PAPM-PROBMIV 2001, Aachen. Vol. 2165 of LNCS. 2001. pp. 184–199. doi:10.1007/3-540-44804-7_12.
- [30] C. Camacho, L. Llana y A. Núñez. Cost-related interface for software product lines. Journal of Logical and Algebraic Methods in Programming 85 (2016) 227–244. doi:10.1016/j.jlamp.2015.09.009.
- [31] C. Camacho, L. Llana, A. Núñez y M. Núñez. Probabilistic software product lines.
- [32] A. Classen, Q. Boucher y P. Heymans. A text-based approach to feature modelling: Syntax and semantics of TVL. Science of Computer Programming 76 (12) (2011) 1130–1143. doi:10.1016/j.scico.2010.10.005.
- [33] P.C. Clements y L. Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.
- [34] K. Czarnecki y S. Helsen. Feature-based survey of model transformation approaches. IBM Systems Journal 45 (3) (2006) 621–646. doi:10.1147/sj.453.0621.

- [35] K. Czarnecki y K. Pietroszek. Verifying feature-based model templates against well-formedness ocl constraints. En: Proceedings of the 5th International Conference on Generative Programming and Component Engineering. GPCE '06. ACM, New York, NY, USA, 2006. pp. 211–220. doi:10.1145/1173706.1173738.
- [36] K. Czarnecki y A. Wasowski. Feature diagrams and logics: There and back again. En: 11th International Software Product Line Conference, SPLC'07. IEEE Computer Society Press, 2007. pp. 23–34. doi:10.1109/SPLC.2007.19.
- [37] C. Dania. Análisis de refinamientos entre sistemas de transiciones modales basado en sat. B.S. Thesis. Universidad Nacional de Cordoba (2008).
- [38] M. Eriksson, J. Borstler y K. Borg. The pluss approach - domain modeling with features, use cases and use case realizations. En: 9th International Conference on Software Product Lines, SPLC'06. Springer-Verlag, 2006. pp. 33–44. doi:10.1007/11554844_5.
- [39] A. Fantechi y S. Gnesi. A behavioural model for product families. En: 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers, ESEC-FSE companion '07. ACM Press, 2007. pp. 521–524. doi:10.1145/1287624.1287700.
- [40] A. Fantechi y S. Gnesi. Formal modeling for product families engineering. En: 12th International Software Product Line Conference, SPLC'08. IEEE Computer Society Press, 2008. pp. 193–202. doi:10.1109/SPLC.2008.45.
- [41] D. Fischbein, S. Uchitel y V. Braberman. A foundation for behavioural conformance in software product line architectures. En: Workshop on Role of software architecture for testing and analysis, ROSATEA'06. ACM Press, 2006. pp. 39–48. doi:10.1145/1147249.1147254.
- [42] C. Gencel. How to use cosmic functional size in effort estimation models?. En: Software Process and Product Measurement. Vol. 5338 of LNCS. Springer Berlin Heidelberg, 2008. pp. 196–207. doi:10.1007/978-3-540-89403-2_17.

- [43] R. Gheyi, T. Massoni y P. Borba. A theory for feature models in Alloy. En: First Alloy Workshop. 2006.
- [44] C. Ghezzi, M. Jazayeri y D. Mandrioli. Fundamentals of Software Engineering. 2nd Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.
- [45] P. Godefroid, M. Huth y R. Jagadeesan. Abstraction-based model checking using modal transition systems. En: 12th International Conference on Concurrency Theory, CONCUR '01. Springer, 2001. pp. 426–440. doi:10.1007/3-540-44685-0_29.
- [46] A. Gruler, M. Leucker y K. Scheidemann. Calculating and modeling common parts of software product lines. En: 12th International Software Product Line Conference, SPLC'08. IEEE Computer Society Press, 2008. pp. 203–212. doi:10.1109/SPLC.2008.22.
- [47] A. Gruler, M. Leucker y K.D. Scheidemann. Modeling and model checking software product lines. En: 10th IFIP WG 6.1 International Conference, FMOODS'08, LNCS 5051. Springer, 2008. pp. 113–131. doi:10.1007/978-3-540-68863-1_8.
- [48] M. Harsu. A survey on domain engineering. Tech. rep.. Institute of Software Systems, Tampere University of Technology (2002).
- [49] P. Heymans, P.Y. Schobbens, J.C. Trigaux, Y. Bontemps, R. Matulevicius y A. Classen. Evaluating formal properties of feature diagram languages. IET Software 2 (3) (2008) 281–302. doi:10.1049/iet-sen:20070055.
- [50] R.M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luetzgen, A.J.H. Simons, S. Vilkomir, M.R. Woodward y H. Zedan. Using formal specifications to support testing. ACM Computing Surveys 41 (2) (2009) 9:1–9:76. doi:10.1145/1459352.1459354.
- [51] J. Hillston. Process algebras for quantitative analysis. En: 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05). 2005. pp. 239–248. doi:10.1109/LICS.2005.35.

- [52] P. Höfner, R. Khédri y B. Möller. Feature algebra. En: 14th International Symposium on Formal Methods, FM'06. Vol. 4085 of LNCS. Springer, 2006. pp. 300–315. doi: 10.1007/11813040_21.
- [53] P. Höfner, R. Khédri y B. Möller. An algebra of product families. *Software and System Modeling* 10 (2) (2011) 161–182. doi:10.1007/s10270-009-0127-2.
- [54] Y. Hold-Geoffroy, O. Gagnon y M. Parizeau. Once you scoop, no need to fork. En: *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014. p. 60. doi:10.1145/2616498.2616565.
- [55] I. Hussain, L. Kosseim y O. Ormandjieva. Approximation of cosmic functional size to support early effort estimation in agile. *Data Knowl. Eng.* 85 (2013) 2–14. doi: 10.1016/j.datak.2012.06.005.
- [56] O. Ibarra y C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM* 22 (4) (1975) 463–468. doi:10.1145/321906.321909.
- [57] M. Janota y G. Botterweck. Formal approach to integrating feature and architecture models. En: 11th International Conference on Fundamental approaches to software engineering, FASE'08, LNCS 4961. Springer, 2008. pp. 31–45. doi:10.1007/978-3-540-78743-3_3.
- [58] M. Jorgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software* 70 (1-2) (2004) 37–60. doi:10.1016/S0164-1212(02)00156-5.
- [59] T. Kahsai, M. Roggenbach y B. Schlingloff. Specification-based testing for refinement. En: 5th IEEE International Conference on Software Engineering and Formal Methods, SEFM'07. IEEE Computer Society Press, 2007. pp. 237–246. doi:10.1109/SEFM.2007.38.
- [60] T. Käkölä y J. C. Dueñas. *Software Product Lines - Research Issues in Engineering and Management*. Springer, 2006. doi:10.1007/978-3-540-33253-4.

- [61] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak y A.S. Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21. Carnegie Mellon University (1990).
- [62] S. Kang, J. Lee, M. Kim y W. Lee. Towards a formal framework for product line test development. En: 7th IEEE International Conference on Computer and Information Technology, CIT'07. IEEE Computer Society Press, 2007. pp. 921–926. doi:10.1109/CIT.2007.40.
- [63] G. Keller y T. Teufel. Sap R/3 Process Oriented Implementation. 1st Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.
- [64] H. Kiyoto y S. Nishizaki. Extended process algebra for cost analysis. Foundations of Computer Science and Technology 2 (2) (2012) 197–214. doi:10.5121/ijfcst.2012.2201.
- [65] K.Pohl, G. Böckle y F.J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer, 2005. doi:10.1007/3-540-28901-1.
- [66] P. Kruchten. Architectural blueprints: The 4+1 view model of software architecture. IEEE Software 12 (6) (1995) 42–50. doi:10.1109/52.469759.
- [67] C. Krueger. Systems and software product line engineering with the spl lifecycle framework. En: Proceedings of the 13th International Software Product Line Conference. SPLC '09. 2009. pp. 307–307. doi:10.1007/978-3-642-15579-6_55.
- [68] C. Krzysztof, H. Simon y W.E. Ulrich. Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice 10 (2) (2005) 143–169. doi:10.1002/spip.225.
- [69] K.G. Larsen, U. Nyman y A. Wasowski. Modal I/O automata for interface and product line theories. En: 16th European conference on Programming, ESOP'07. Springer, 2007. pp. 64–79. doi:10.1007/978-3-540-71316-6_6.

- [70] K.G Larsen, U. Nyman y A. Wasowski. Modeling software product lines using color-blind transition systems. *International J. Softw. Tools Technol. Transf.* 9 (2007) 471–487. doi:10.1007/s10009-007-0046-x.
- [71] K. Larsen y A. Skou. Bisimulation through probabilistic testing. *Information and Computation* 94 (1) (1991) 1 – 28. doi:10.1016/0890-5401(91)90030-6.
- [72] K.G. Larsen, B. Steffen y C. Weise. A constraint oriented proof methodology based on modal transition systems. En: *1st International Workshop on Tools and Algorithms for Construction and Analysis of Systems*. Springer, 1995. pp. 17–40. doi:10.1007/3-540-60630-0_2.
- [73] K. G. Larsen, U. Nyman y A. Wasowski. On modal refinement and consistency. En: *18th International Conference on Concurrency Theory, CONCUR’07, LNCS 4703*. 2007. pp. 105–119. doi:10.1007/978-3-540-74407-8_8.
- [74] K. Guldstrand Larsen y B. Thomsen. A modal process logic. En: *Logic in Computer Science*. 1988. doi:10.1109/LICS.1988.5119.
- [75] M. Mannion. Using first-order logic for product line model validation. En: *2nd International Software Product Line Conference, SPLC’02*. Springer, 2002. pp. 176–187. doi:10.1007/3-540-45652-X_11.
- [76] J.D. McGregor. Testing a software product line. Tech. Rep. CMU/SEI-2001-TR-022. Carnegie Mellon University, Software Engineering Institute (2001).
- [77] J.D. McGregor, P. Sodhani y S. Madhavapeddi. Testing variability in a software product line. En: *SPLiT – Workshop on Software Product Line Testing*. 2004. doi:10.1007/978-3-540-28630-1_37.
- [78] M. Mendonça, A. Wasowski y K. Czarnecki. SAT-based analysis of feature models is easy. En: *13rd International Software Product Line Conference, SPLC’09*. 2009. pp. 231–240. doi:10.1145/1753235.1753267.

- [79] R. Milner. A Calculus of Communicating Systems (LNCS 92). Springer, 1980. doi:10.1007/3-540-10235-3.
- [80] M. Roggenbach. Csp-casl - a new integration of process algebra and algebraic specification. Theoretical Computer Science 354 (1) (2006) 42–71. doi:10.1016/j.tcs.2005.11.007.
- [81] R. Muschevici, J. Proença y D. Clarke. Feature nets: behavioural modelling of software product lines. Software & Systems Modeling 15 (4) (2016) 1181–1206. doi:10.1007/s10270-015-0475-z.
- [82] V. Nguyen, S. Deeds-rubin, T. Tan y B. Boehm. A sloc counting standard. En: CO-COMO II Forum 2007. 2007. pp. 1–16.
- [83] S. Nishizaki y H. Kiyoto. Formal framework for cost analysis based on process algebra. En: Maotai Zhao y Junpin Sha (Eds.), Communications and Information Processing. Vol. 288 of Communications in Computer and Information Science. Springer Berlin Heidelberg, 2012. pp. 110–117. doi:10.1007/978-3-642-31965-5_13.
- [84] J. Nummenmaa, T. Nummenmaa y Z. Zhang. On the use of ltss to analyze software product line products composed of features. En: Fuchun Sun, Tianrui Li y Hongbo Li (Eds.), Knowledge Engineering and Management. Vol. 214 of Advances in Intelligent Systems and Computing. Springer Berlin Heidelberg, 2014. pp. 531–541. doi:10.1007/978-3-642-37832-4_48.
- [85] A. Núñez y R. Hierons. A methodology for validating cloud models using metamorphic testing. annals of telecommunications - annales des télécommunications 70 (3) (2015) 127–135. doi:10.1007/s12243-014-0442-7.
- [86] U. Nyman. Modal transition systems as the basis for interface theories and product lines. Ph.D. thesis. Aalborg University (2008).
- [87] A. Oberweis, V. Pankratius y W. Stucky. Product lines for digital information products. Information Systems 32 (6) (2007) 909–939. doi:10.1016/j.is.2006.09.003.

- [88] A. Page y K. Johnston. *How We Test Software at Microsoft*. Microsoft Press, 2008.
- [89] D. Perry y A. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes* 17 (4) (1992) 40–52. doi:10.1145/141874.141884.
- [90] S. Reis, A. Metzger y K. Pohl. Integration testing in software product line engineering: a model-based technique. En: *10th International Conference on Fundamental approaches to software engineering, FASE'07*. Springer, 2007. pp. 321–335. doi:10.1007/978-3-540-71289-3_25.
- [91] A. Reuys, E. Kamsties, K. Pohl y S. Reis. Model-based system testing of software product families. En: *17th International Conference on Advanced Information Systems Engineering, CASEI'05, LNCS 3520*. Vol. 3520. Springer, 2005. pp. 519–534. doi:10.1007/11431855_36.
- [92] M. Riebisch, K. Böllert, D. Streitferdt y I. Philippow. Extending feature diagrams with UML multiplicities. En: *6th World Conference on Integrated Design & Process Technology, IDPT'02*. 2002.
- [93] T. Satyananda, D. Lee, S. Kang y S. Hashmi. Identifying traceability between feature model and software architecture in software product line using formal concept analysis. En: *Proceedings of the The 2007 International Conference Computational Science and Its Applications. ICCSA '07*. IEEE Computer Society, Washington, DC, USA, 2007. pp. 380–388. doi:10.1109/ICCSA.2007.47.
- [94] H. Schackmann y H. Horst. A cost-based approach to software product line management. En: *Proceedings of the International Workshop on Software Product Management. IWSPM '06*. IEEE Computer Society Press, 2006. pp. 13–18. doi:10.1109/IWSPM.2006.1.
- [95] P. Y. Schobbens, P. Heymans, J.-C. Trigaux y Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks* 51 (2) (2007) 456–479. doi:10.1016/j.comnet.2006.08.008.

- [96] S. Segura, R.M. Hierons, D. Benavides y A. Ruiz. Automated metamorphic testing on the analyses of feature models. *Information & Software Technology* 53 (3) (2011) 245–258. doi:10.1016/j.infsof.2010.11.002.
- [97] T. Simpson, J. Jiao, Z. Siddique y K. Holtta-Otto. *Advances in Product Family and Product Platform Design*. Springer, 2014. doi:10.1007/978-1-4614-7937-6.
- [98] P. Sochos, I. Philippow y M. Riebisch. Feature-oriented development of software product lines: Mapping feature models to the architecture. En: Mathias Weske y Peter Liggesmeyer (Eds.), *Object-Oriented and Internet-Based Technologies*, LNCS 3263. Springer, 2004. pp. 23–42. doi:10.1007/978-3-540-30196-7_11.
- [99] I. Sommerville. *Software Engineering* (5th Ed.). Addison Wesley Longman Publishing Co., Inc., 1995.
- [100] J. Sun, H. Zhang y H. Wang. Formal semantics and verification for feature modeling. En: *10th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS'05*. IEEE Computer Society Press, 2005. pp. 303–312. doi:10.1109/ICECCS.2005.48.
- [101] W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology* 41 (10) (1999) 639 – 650. doi:10.1016/S0950-5849(99)00016-6.
- [102] F.J. van der Linden, K. Schmid y E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2007. doi:10.1007/978-3-540-71437-8.
- [103] R.J. Vanglabbeek, S.A. Smolka y B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation* 121 (1) (1995) 59 – 80. doi:http://dx.doi.org/10.1006/inco.1995.1123.