Universidad Politécnica de Valencia

Departamento de Sistemas
Informáticos y Computación

Tesis de Master

# Técnicas de Reescritura Ecuacional aplicadas al Análisis, Verificación y Transformación de sistemas Web

Candidato:
Pedro Ojeda Forés

Directores:
María Alpuente Frasnedo
Santiago Escobar Román

– Septiembre de 2008 –

Correo Electrónico del autor: pedojfo@gmail.com

Dirección del autor:

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera, s/n
46022 Valencia
España

*...a mis padres y a mi esposa*

# Agradecimientos

En primer lugar, quiero agradecer a mis directores María Alpuente Frasnedo y Santiago Escobar Román y a mi compañero y amigo Daniel Omar Romero, todo el apoyo y dedicación que me han dado desde el día que entre a formar parte del grupo ELP, y que sin su ayuda no hubiera podido realizar este trabajo.

A mis padres, por hacer que este momento llegue gracias al esfuerzo que siempre han realizado para que pueda realizar mis estudios.

Quiero agradecer a mi familia, en especial a mis padres Pedro y Carmen, a mis hermanos Carmen y Joan, a mi cuñado Jorge, a mis otros padres Juan y Pepita, a mi abuelo Amador y a mi mujer Lorena, todo el apoyo, confianza, ánimos y cariño que siempre recibo por su parte.

A mis amigos, en especial a Alex, Isabel, Dani, Carlos, Julio, Diego, Martina, Sonia, Ana, Aristides, que me han entregado su amor y apoyo siempre.

A José Meseguer y Moreno Falaschi, por la confianza depositada en mi.

Por último quiero dar gracias a los miembros del grupo ELP y a todas las personas que siempre han creído en mi.

# Prefacio

En esta tesis se presenta en primer lugar, un marco para la verificación y reparación de sitios web. En esta primera parte de la tesis, proporcionamos un marco de verificación que nos permite definir propiedades útiles para poder comprobar si un sitio web es correcto en base a una especificación. Una vez analizado el sitio web con respecto a la especificación, nuestra técnica nos indica si existen datos incorrectos y si hay páginas en el sitio web que carecen de información o bien no existen tales páginas. El marco de verificación ha sido implementado el lenguaje *Maude* en el prototipo *WebVerdi-M* mejorando considerablemente una implementación anterior de la técnica propuesta.

La segunda parte de esta tesis vino motivada por nuestra intención de mejorar las técnicas de diagnostico de errores del prototipo *WebVerdi-M*. En concreto, el cálculo de puntos fijos que es central al motor Maude de la herramienta obedece condiciones de parada que se basan en un cálculo de generalizaciones. Con objeto de enriquecer dicho cálculo considerando que las funciones puedan obedecer axiomas ecuacionales como asociatividad, conmutatividad e identidad, acometimos el diseño de un algoritmo para el cálculo de generalizaciones ecuacionales, y *order-sorted*. El prototipo implementado se utiliza actualmente ya en otras herramientas desarrolladas en el grupo ELP (en concreto en el analizador de protocolos NPA). En el futuro está previsto el desarrollo de una evaluador parcial para Maude, para cuya terminación será fundamental el sistema de inferencia que calcula lggs ecuacionales y que ha sido desarrollado en esta tesis de master.

# Índice general

# Índice de figuras

# Parte I

# Verificación y Reparación de Sitios Web

# Resumen de Contribuciones de la Parte I

Con el crecimiento explosivo de las "tecnologías de la información", la distinción entre "sistemas críticos" e "informática de consumo" se ha reducido drásticamente, por lo que la ausencia de garantías de calidad del software resulta inadmisible hoy en día en todos los ámbitos y no sólo ya en aquellos dominios donde las consecuencias de un fallo resultaban tradicionalmente críticas.

Para hacer posible la deseable transferencia tecnológica a la industria de los resultados de la investigación en métodos formales para la especificación, verificación y reparación de sistemas de software, es necesario reducir drásticamente los costes de la especificación y automatizar completamente los análisis. Esto es particularmente necesario en el nuevo escenario que supone la Web, donde la proliferación de datos, aplicaciones y servicios está creando desafíos específicos y urgentes.

En este trabajo se presenta en primer lugar una metodología y sistema experimental de verificación y reparación de sitios Web. Nuestro marco permite especificar los requerimientos del usuario sobre el sitio Web, detectar con precisión las partes que no satisfacen la especificación y aplicar estrategias de reparación para obtener un sitio que sea correcto y completo con respecto a la especificación inicial. Brevemente, las principales contribuciones en esta parte de la tesis son:

- Definimos estrategias de reparación que optimizan el proceso de reparación de sitios Web.
- Realizamos un estudio de escalabilidad (en función del tiempo de ejecución y tamaño del sitio Web) de la metodología de verificación descrita en [Alpuente et al., 2006a].
- Presentamos, de acuerdo a nuestro conocimiento, la primera metodología abstracta de verificación formal de sitios Web que soporta aspectos estáticos

así como dinámicos.

- Describimos una arquitectura orientada a servicios que hace fácilmente accesibles las capacidades de verificación Web propuestas.

# 1

# Introducción

Con el auge espectacular de las tecnologías de la información y su impacto en la sociedad, la necesidad de disponer de herramientas que hagan más fácil el análisis y la verificación de la calidad de productos software no es cuestionado hoy en día por nadie. Por ello, numerosos grupos de investigación en todo el mundo centran sus esfuerzos en el diseño y desarrollo de todo tipo formalismos y herramientas con ese mismo objetivo final: garantizar que al usuario (en el sentido más amplio de la palabra) se le proporcione un producto de calidad.

Los sistemas software para la web se han convertido en un instrumento insustituible de la moderna sociedad de la información: hacen posible el intercambio de información de forma rápida y a escala global; constituyen el medio natural de las grandes transacciones financieras; permiten acceder de forma rápida y selectiva a grandes volúmenes de información especializada, etc. Todo esto ha incrementado la complejidad de los sitios web y puesto de manifiesto la necesidad de asistir a los administradores de sistemas web en la detección y reparación de las posibles incorrecciones o inconsistencias.

En este escenario, es esencial el desarrollo de métodos, modelos y herramientas que se apliquen a la verificación formal de sitios y servicios web, y que permitan no sólo detectar posibles errores en los enlaces o en la estructura sino también en la semántica de los mismos. Los fallos de calidad deben ser detectados con precisión para que, de esta manera, se puedan aplicar estrategias de reparación (idealmente) automáticas que permitan obtener sitios web que sean correctos y completos con respecto a una especificación o modelo conceptual de los mismos.

## 1.1. Estado del arte

El concepto de Sitio Web (*Web Site*) contiene una ambigüedad oculta, ¿es una entidad estática, para ser vista como colección de datos, o es una entidad dinámica, para ser vista según la concepción que tienen de él los usuarios?. Esta distinción impacta en la manera en la cuál se analiza y verifica un sitio Web. Un análisis estático de flujo de datos puede aproximar el comportamiento del sistema, pero usualmente es menos específico; un análisis dinámico puede solamente ofrecer garantías relativas al comportamiento de estos examinados, pero la estructura de la información contextual adicional puede a menudo ofrecer más respuestas informativas. Esta distinción se hace mas compleja en la Web debido a la naturaleza de las interacciones Web [Shiriram, 2005].

Una situación característica que se presenta en las aplicaciones Web es la del uso de lenguajes con *script* que, al ser lenguajes interpretados por el servidor, éste envía al navegador el código *HTML* resultante. En [Stone, 2005] se presenta un prototipo escrito en PHP de una herramienta para validar documentos que contienen *script*, la cual genera una expresión que será validada respecto a un *DTD* extendido. Esta herramienta presenta limitaciones en la sintaxis de los comandos dentro de los *script* y además depende del lenguaje considerado.

En este trabajo es de nuestro interés el uso de técnicas de reescritura para la definición y verificación de las propiedades de un sitio Web. La investigación en este ámbito es muy reciente. En [Lucas, 2005] se muestra cómo las técnicas de reescritura se pueden utilizar para modelar el comportamiento dinámico de sitios Web, usando Maude como lenguaje de especificación. El artículo explora la propiedad de alcanzabilidad dentro de un sitio Web, dando la posibilidad de obtener un modelo más expresivo y/o analizar otros comportamientos como: estructura de los nombres, eficiencia de caminos del buscador, frecuencia de uso de los diferentes enlaces, etc. Finalmente, se apuntan otros beneficios posibles del uso de técnicas de reescritura que pueden ser futuros temas de investigación para el análisis de la Web, como la definición estructurada de sitios Web y la evolución de los mismos.

Otros trabajos [Alpuente et al., 2006a; Alpuente et al., 2007a; Alpuente et al., 2007e; Ballis, 2005; Ballis and Vivó, 2005] utilizan reescritura en este ámbito. En [Alpuente et al., 2006a; Ballis, 2005] se desarrolla un marco para la verificación

automática de sitios Web que puede ser usado para especificar condiciones de integridad dadas de un sitio y verificar automáticamente si se cumplen. Este marco está implementado en el prototipo del sistema de verificación WebVerdi-M [Alpuente et al., 2007a; Alpuente et al., 2007e]. El entorno proporciona un lenguaje de especificación para la definición de las propiedades semánticas del sitio Web y, mediante una técnica de verificación formal, se obtienen los requerimientos que no se satisfacen. La metodología está basada en una técnica de reescritura no estándar, en la cual se reemplaza el ajuste de patrones tradicional por un mecanismo de simulación de árboles, que resulta adecuada como base para formular una técnica de reconocimiento de patrones dentro de un documento semiestructurado.

Otro tipo de trabajos se orienta a verificar el código *HTML* de los sitios Web. Por ejemplo, en [Kirchner et al., 2005] se desarrolló la idea de anclaje modular en el lenguaje *HTML*, introduciendo primitivas simples para importación y parametrización, y definiendo una extensión modular de *HTML*.

Otro escenario de verificación en la Web se presenta cuando se trabaja con servicios Web que interoperan en ambientes de red heterogéneos. En este caso, la mayoría de los ambientes de verificación han sido construidos para arquitecturas tradicionales. En [Ferrari et al., 2004] se presenta el prototipo de una herramienta que explota el paradigma arquitectónico de los servicios Web distribuidos. En [Brogi et al., 2004] se muestra como la descripción WSCI *(Web Service Choreography Interface)* puede ser formalizada usando el álgebra de procesos CCS. De esta manera, verificar si dos o más servicios Web son compatibles para interoperar o no. En caso de no serlo, se generara cuando es posible de manera automática, la especificación del adaptador que media entre ellos.

En [Haydar et al., 2004] se propone el uso de autómatas para representar los estados de un sitio Web, modelando las aplicaciones Web mediante autómatas finitos comunicados, que se construyen según las propiedades definidas a verificar. El método genera el autómata de manera automática durante la sesión del navegador y utiliza técnicas de verificación algorítmica *(model cheking)* para verificar las propiedades de interés.

En el marco de la reparación existen también diferentes líneas de investigación incipientes. En [Nentwich et al., 2003], se presenta un marco para reparar inconsistencias en documentos distribuidos, como un complemento a la herramienta

xlinkit [Capra et al., 2002]. La principal contribución es la semántica que pone en correspondencia el lenguaje lógico de primer orden de xlinkit a un catálogo de acciones reparadoras que pueden ser usadas para corregir de forma interactiva las violaciones de reglas, aunque este trabajo no predice si la ejecución de la acción puede provocar una nueva violación a dicha reglas. Tampoco es posible detectar si dos expresiones que formulan un requerimiento para el sitio Web son incompatibles. Similarmente, en [Scheffczyk et al., 2004a; Scheffczyk et al., 2004b] se presenta una extensión para CDET [Scheffczyk et al., 2003]. Esta extensión incluye un mecanismo para eliminar inconsistencias de un conjunto de documentos interrelacionados. Primero se genera un grafo dirigido acíclico ($DAG$) que representa las relaciones entre documentos, y entonces las reparaciones se deriban directamente de este grafo. En este caso, las reglas temporales soportan interferencias y la compatibilidad de las reparaciones es tenida en cuenta explícitamente. Desafortunadamente, esta compatibilidad resulta muy costosa de chequear en reglas temporales. Ambas aproximaciones se basan en técnicas del campo de las bases de datos activas [Bertossi and Pinto, 1999]. Otras investigaciones recientes en este campo se centran en la derivación de reglas activas que disparen, de manera automática, acciones reparadoras que conducen a un estado consistente después de cada actualización [Mayol and Teniente, 1999].

Esta lista no exhaustiva de trabajos constituye una panorámica de las principales líneas de investigación existentes relacionadas con la verificación y reparación formal de sitios Web, incluyendo una visión inicial del estado del arte de nuestra propuesta.

## 1.2. Contribuciones de la tesis

En [Alpuente et al., 2006a; Ballis, 2005], se presenta un marco para la verificación automática de sitios Web. Dicho marco permite especificar restricciones de integridad para un sitio Web y entonces comprobar automáticamente si estas restricciones se satisfacen. El marco proporciona un lenguaje de especificación que permite definir propiedades sintácticas así como semánticas de un sitio Web. Como resultado de la verificación se identifican dos tipos de errores: errores de corrección (*correctness errors*) y errores de completitud (*completeness errors*). La

metodología propuesta se implementó en el prototipo Verdi [Ballis, 2005; Ballis and Vivó, 2005], desarrollado inicialmente en Haskell. Una vez realizada la verificación resulta interesante poder corregir las anomalías encontradas.

Las nuevas contribuciones originales que ahora se presentan y que configuran el cuerpo de los capítulos de esta tesis dedicados a la verificación y reparación de la web son las siguientes.

**Un motor algebráico para la verificación de sistemas web en GVerdi** (ver Capítulo 4) M. Alpuente, D. Ballis, S. Escobar, M. Falaschi, P. Ojeda, y D. Romero. *Un motor algebráico para la verificación de sistemas web en GVerdi*. Technical Report. Departamento de Sistemas Informáticos y computación. DSIC-II/02/07.

En este trabajo, describimos un nuevo motor para el sistema de verificación *GVerdi* [Ballis, 2005; Ballis and Vivó, 2005] desarrollado en *Maude* [Clavel et al., 2007] así como la extensión del sistema *GVerdi* para poder interactuar con el nuevo motor, que aporta mejoras significativas.

**Estrategias de reparación para sitios Web incompletos** (ver Capítulo 5.5) **Trabajo.** M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero. *Estrategias de Reparación para Sitios Web Incompletos*. XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007). Octubre 1–5, 2007 – Corrientes, Argentina.

En este trabajo, completamos la metodología de reparación semiautomática de sitios Web erróneos presentada en [Alpuente et al., 2006b] con el tratamiento y eliminación de los errores de completitud. Para ello, formalizamos dos relaciones de órden parcial ($\preceq_{inf}$ y $\preceq^{sup}$) sobre los errores de completitud detectados. Con estos órdenes, realizamos un análisis de la dependencia existente entre los errores, lo cual nos permite definir dos estrategias de reparación: *reduce-delete-actions* y *reduce-insertion-actions*. La primera se focaliza en la reducción de la cantidad de información que se necesita eliminar para obtener un sitio Web libre de errores con respecto a su especificación. La segunda minimiza la información que se necesita insertar. Ambas estrategias explotan la dependencia existente entre los errores, reparando todos sin más que actuar explícitamente sobre un subconjunto del total

de ellos.

La consideración de estas estrategias nos lleva a una optimización efectiva del proceso de reparación de un sitio Web, debido a que el usuario debe de reparar una cantidad menor de errores para obtener un sitio Web correcto y completo.

**Un marco genérico abstracto para la verificación de sitios Web** (ver Capítulo 6)

**Trabajo.** M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero. *An Abstract Generic Framework for Web Sites Verification.* The IEEE/IPSJ Symposium on Applications and the Internet, (SAINT 2008). Julio 28, 2008 – Turku, Finland. IEEE Computer Society.

La *interpretación abstracta* es una teoría que permite extraer información relevante de programas sin considerar todos los detalles dados por la semántica estándar. De acuerdo con nuestro conocimiento, este trabajo presenta la primera metodología abstracta que soporta la verificación de aspectos estáticos así como dinámicos de sitios Web.

En [Alpuente et al., 2007b; Alpuente et al., 2007d], hicimos un estudio de escalabilidad (en función del tiempo de ejecución y tamaño del sitio Web) de la metodología de verificación Web. Con respecto a la verificación de la completitud, detectamos que el tiempo invertido en la computación del punto fijo que se requiere para la evaluación de las reglas de completiud, excede el tiempo deseado. Por esto, aquí describimos un trabajo concerniente a una metodología abstracta para el análisis y verificación de sitios Web que permite compensar el alto costo de ejecución de analizar sitios Web complejos. En nuestro trabajo, la abstracción se formaliza como una transformación *source-to-source* (fuente a fuente) que es paramétrica con respecto al dominio abstracto considerado, lo que permite una traducción de los documentos Web y las reglas de la especificación en construcciones del lenguaje original. De esta manera, resulta inmediato derivar una implementación con poco esfuerzo. La idea clave en la abstracción es explotar la similitud que existe entre diferentes sub-estructuras que son comúnmente encontradas en los documentos HTML/XML.

**Implementación y resultados experimentales** (ver Capítulo 7)

**Trabajo.** M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero. *A Fast Algebraic Web Verification Service.* Proc. of First Int'l Conf. on Web Reasoning and Rule Systems (RR 2007) Pages 239 – 248. Junio 2007 – Innsbruck, Austria. Lecture Note in Computer Science 4524, páginas 239–248. Springer-Verlag Berlin Heidelberg 2007.

**Trabajo.** M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero. *The Web Verification Service WebVerdi-M.* VII Jornadas sobre PROgramación y LEnguajes (PROLE 2007), en el marco del Congreso Español de Informática (CEDI 2007). Septiembre 12–14, 2007 – Zaragoza, España.

El objetivo de estos trabajos fue evaluar la utilidad de nuestro sistema en escenarios de grandes volúmenes de datos, para ello utilizamos el generador de documentos XML xmlgen [Centrum voor Wiskunde en Informatica, 2001].

Comenzamos este trabajo investigando las capacidades de Maude que son convenientes para nuestra implementación, tales como: *AC pattern-maching* y metaprogramación. Con estas capacidades, proveemos a WebVerdi-M con un poderoso motor de verificación Web. Luego, como un contenido importante de estos trabajos, presentamos los resultados obtenidos en las comparaciones del tiempo de ejecución del motor Verdi-M respecto a la implementación previa [Ballis and Vivó, 2005]. Los resultados obtenidos son muy prometedores y muestran excelentes prestaciones (por ejemplo, menos de un segundo para evaluar un árbol de aproximadamente 30.000 nodos). Finalmente, describimos una arquitectura orientada a servicios que hace fácilmente accesible (para los usuarios de Internet) las capacidades de verificación Web de nuestro sistema.

Un factor importante en nuestro diseño del sistema ha sido el reducir el coste de aprendizaje, motivo por el cual desarrollamos una interfaz amigable de usuario. El prototipo desarrollado WebVerdi-M está publicamente disponible junto con un conjunto de ejemplos y su API XML en

`http://www.dsic.upv.es/users/elp/webverdi-m/`.

## 1.3. Estructura

El resto de esta parte del trabajo está organizado como sigue. En el Capítulo 2, se resumen algunos conceptos preliminares. En el Capítulo 3, recordaremos brevemente la metodología de verificación formal. En el Capítulo 5 describimos la metodología de reparación y las estrategias para optimizar la reparación de los errores de completitud. En el Capítulo 6, presentamos, de acuerdo con nuestro conocimiento, la primera metodología abstracta que considera la verificación de aspectos estáticos así como dinámicos de sitios Web. En el Capítulo 7, describimos la implementación de nuestro prototipo WebVerdi-M: en primer lugar, recordamos las principales características del lenguaje Maude que fueron explotadas para la implementación de nuestro motor de verificación de sitios Web. A continuación, describimos la arquitectura del prototipo WebVerdi-M y los resultados experimentales obtenidos al utilizar WebVerdi-M con grandes volúmenes de datos. Por último, en el Capítulo 8 describimos las conclusiones de esta parte de la tesis.

# 2

# Preliminares

En esta sección, recordamos algunas definiciones preliminares junto con la terminología necesaria para el entendimiento de esta parte del trabajo.

Decimos que un conjunto finito de símbolos es un *alfabeto*. Dado el alfabeto $A$, $A^*$ denota el conjunto de todas las secuencias finitas de elementos sobre $A$. La igualdad sintáctica entre objetos se representa como $\equiv$.

Denotamos por $\mathcal{V}$ un conjunto infinito de variables y $\Sigma$ denota un conjunto de símbolos de función, o *signatura*. Consideramos las signaturas de aridad variable como en [Dershowitz and Plaisted., 2001] (i.e., signaturas en las cuales los símbolos tienen aridad no especificada, es decir, pueden tener cualquier número de argumentos a continuación). $\tau(\Sigma, \mathcal{V})$ y $\tau(\Sigma)$ denotan el *álgebra de términos con variables* y el *álgebra de términos básicos* construidas en $\Sigma \cup \mathcal{V}$ y $\Sigma$, respectivamente.

Las posiciones son representadas por secuencias de números naturales que denotan el camino de acceso a un término. La secuencia vacía $\Lambda$ denota la posición raíz. Con la notación $w_1.w_2$, se denota la concatenación de la posición $w_1$ y la posición $w_2$. Las posiciones son ordenadas por el orden de prefijos, que es, dada las posiciones $w_1, w_2$, $w_1 \leq w_2$ si existe una posición $x$ s.t. $w_1.x = w_2$.

Dado $S \subseteq \Sigma \cup \mathcal{V}$, $O_S(t)$ denota el conjunto de posiciones de un término $t$ que tenga como raíz al símbolo $S$. Por otra parte, para una posición $x$, $\{x\}.O_S(t) = \{x.w \mid w \in O_S(t)\}$.

Con $t|_v$ representamos el subtérmino cuya raíz es $v$ de $t$. $t[r]_v$ es el término $t$ con el subtérmino cuya raíz es $v$ sustituido por el término $r$.

Una *sustitución* $\sigma \equiv \{X_1/t_1, X_2/t_2, \ldots\}$ es una aplicación del conjunto de variables $\mathcal{V}$ en el conjunto de términos $\tau(\Sigma, \mathcal{V})$ que satisface las siguientes condi-

ciones: $(i)X_i \neq X_j$, si $i \neq j$, $(ii)X_i\sigma = t_i, i = 1,..n$, y $(iii)X\sigma = X$, para $X \in \mathcal{V} \setminus \{X_1, ..., X_n\}$. Con $\varepsilon$ representamos una *sustitución vacía*. Dada una sustitución $\sigma$, el *dominio* de $\sigma$ es el conjunto $Dom(\sigma) = \{X | X\sigma \not\equiv X\}$. Dadas las sustituciones $\sigma_1$ y $\sigma_2$, tal que $Dom(\sigma_1) \subseteq Dom(\sigma_2)$, por $\sigma_1/\sigma_2$, definimos la sustitución $\{X/t \in \sigma_1 | X \in Dom(\sigma_1) \setminus Dom(\sigma_2)\} \cup \{X/t \in \sigma_2 | X \in Dom(\sigma_1) \cap Dom(\sigma_2)\} \cup \{X/X | X \notin Dom(\sigma_1)\}$. Una *instancia* de un término $t$ se define como $t\sigma$, donde $\sigma$ es una sustitución. Con $Var(s)$ representamos el conjunto de variables que aparecen en el objeto sintáctico $s$.

Los sistemas de rescritura de términos proporcionan un modelo computacional adecuado para los lenguajes funcionales. En consecuencia, seguimos un marco estándar de rescritura de términos para formalizar nuestra propuesta (ver [Baader and Nipkow, 1998; Klop, 1992]).

Un *sistema de rescritura de términos* (TRS para abreviar) es un par $(\Sigma, R)$, donde $\Sigma$ es una signatura y $R$ es un conjunto finito de reglas de reducciones (o rescrituras) de la forma $\lambda \to \rho, \lambda, \rho \in \tau(\Sigma, \mathcal{V}), \lambda \notin \mathcal{V}$ and $Var(\rho) \subseteq Var(\lambda)$. A menudo escribimos $R$ en vez de $(\Sigma, R)$. Un paso de rescritura es la aplicación de una regla de rescritura a una expresión.

Un término $s$ se rescribe a un término $t$ via $r \in R$, $s \to_r t$ (or $s \to_R t$), si existe una posición $u \in O_\Sigma(s)$, $r \equiv \lambda \to \rho$, y una sustitución $\sigma$ tal que $s_{|u} \equiv \lambda\sigma$ y $t \equiv s[\rho\sigma]_u$. Cuando no haya riesgo de confusión, omitiremos cualquier subíndice (por ejemplo $s \to t$). Un término $s$ está en una *forma irreducible* (o forma normal) w.r.t. $R$ si no existe un término $t$ s.t. $s \to_R t$. Un $t$ es la forma irreducible de $s$ w.r.t. $R$ (en símbolos $s \to_R^! t$) si $s \to_R^* t$ y $t$ es irreducible.

Decimos que un TRS $R$ es *terminante*, si no existe una secuencia infinita $t_1 \to_R t_2 \to_R ....$ Un TRS $R$ es *confluente* si, para todos los términos $s, t_1, t_2$, tal que $s \to_R^* t_1$ y $s \to_R^* t_2$, existe un término $t$ s.t. $t_1 \to_R^* t$ y $t_2 \to_R^* t$. Cuando $R$ es terminante y confluente, se denomina *canónico*. En un TRS canónico, cada término de entrada $t$ puede ser reducido a una única forma *irreducible*.

**Ejemplo 2.0.1** *Sea $R$ el siguiente TRS canónico que define estas tres funciones.*

$$\begin{aligned}
\mathtt{sum}(\mathtt{X}, \mathtt{0}) &\to \mathtt{X} \\
\mathtt{sum}(\mathtt{s}(\mathtt{X}), \mathtt{Y}) &\to \mathtt{s}(\mathtt{sum}(\mathtt{X}, \mathtt{Y})) \\
\mathtt{append}(\mathtt{L_1}, [\,]) &\to \mathtt{L_1} \\
\mathtt{append}([\mathtt{X}|\mathtt{L_1}], \mathtt{L_2}) &\to [\mathtt{X}|\mathtt{append}(\mathtt{L_1}, \mathtt{L_2})] \\
\leq (\mathtt{0}, \mathtt{X}) &\to \mathtt{true} \\
\leq (\mathtt{s}(\mathtt{X}), \mathtt{0}) &\to \mathtt{false} \\
\leq (\mathtt{s}(\mathtt{X}), \mathtt{s}(\mathtt{Y})) &\to \leq (\mathtt{X}, \mathtt{Y})
\end{aligned}$$

*La función* sum *calcula la suma de dos números naturales, la función* append *concatena dos listas y la función $\leq$ define la relación "menor o igual que" entre dos números naturales. Los número naturales se representan en notación de Peano por medio de la constante* 0 *y el operador unario* s*. Abusando de la notación, escribiremos $n \in \mathbb{N}$ como abreviatura para $\mathtt{s}^n(\mathtt{0})$. Usamos la notación estándar para las listas, siendo* [ ] *la lista vacía. Los* strings *son listas de caracteres, como de costumbre.*

# 3

# Verificación y reparación de sitios Web

En esta sección recordaremos brevemente en primer lugar la metodología de verificación formal propuesta en [Alpuente et al., 2006a]. A continuación, describimos la metodología de reparación dada en [Alpuente et al., 2006b]. La metodología de verificación [Alpuente et al., 2006a], permite detectar información prohibida como así también faltante dentro de un sitio Web. La ejecución de una especificación Web, tomando como "dato" los documentos que componen un sitio Web, permite renococer y localizar la información que discrepa entre el sitio y las propiedades establecidas por la especificación.

Por otro lado, la metodología de reparación [Alpuente et al., 2006b], permite obtener un sitio web correcto y completo con respecto a su especificación.

En primer lugar, veremos la notación utilizada para denotar un sitio Web y el lenguaje de especificación utilizado. Luego, los conceptos de simulación y reescritura parcial, y finalmente los lineamientos generales de la metodología de verificación y reparación.

## 3.1. Denotación de sitios Web

En [Alpuente et al., 2006a], una página Web puede ser un documento XML [World Wide Web Consortium, 1999] o un documento XHTML [World Wide Web Consortium, 2000], los cuales asumiremos que están bien formados, debido a que existen en la actualidad sistemas software y servicios *on-line* como Tidy [Nesbit, 2000] que son capaces de validar y corregir las sintaxis XHTML/XML, o Doctor

HTML [Imagiware, ], que además incluye comprobación de enlaces.

Sean dos alfabetos $T$ y $\mathcal{T}ag$. Representaremos el conjunto $T^*$ por $\mathcal{T}ext$. Llamaremos a un objeto $\mathtt{t} \in \mathcal{T}ag$ elemento *tag*, y a un objeto $\mathtt{w} \in \mathcal{T}ext$ lo llamaremos elemento *text*. Debido a que las paginas Web se proporcionan con una estructura de árbol o similar, éstas pueden ser traducidas de forma directa a términos ordinarios de un álgebra de términos dada $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$. De esta manera, un sitio Web puede ser representado como un conjunto finito de términos básicos *(ground)*. Nótese que los atributos tag XML/XHTML pueden considerarse elementos etiquetados comunes y, por tanto, traducidos de la misma forma que éstos.

En lo siguiente, también consideraremos los términos del álgebra de términos no básicos *(no-ground)* $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, los cuales pueden contener variables. Un elemento $\mathtt{s} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$ es llamado *Web page template*.

En esta metodología los *Web page templates* son usados para especificar propiedades sobre los sitos Web, como se describe en la Sección 3.2.

**Ejemplo 3.1.1** *La Figura 3.1 representa la traducción de una página Web a términos del álgebra, y la Figura 3.2 un ejemplo de un sitio Web correspondiente a un blog, en donde en la página $p_1$ tenemos dos entradas del blog, $p_2$ y $p_3$ corresponden a las entradas de "culture" y "history" respectivamente. Allí están también los comentarios de cada una de esas entradas. La página $p_4$ es la que contiene los miembros del blog.*

## 3.2. Lenguaje de especificación

De acuerdo con [Alpuente et al., 2006a], una especificación Web es una terna $(I_N, I_M, R)$, donde $I_N$, $I_M$ y $R$ son conjuntos finitos de reglas.

El conjunto $R$ contiene la definición de algunas funciones auxiliares provistas por el usuario tales como procesamiento de cadenas, funciones aritméticas, operadores booleanos, etc. Este conjunto se formaliza como un sistema de reescritura de términos el cual es tratado mediante reescritura estándar [Klop, 1992].

El conjunto $I_N$ describe restricciones para detectar páginas Web erróneas (*reglas de corrección*). Una regla de corrección tiene la siguiente forma: $\mathtt{l} \rightharpoonup \mathtt{error} \mid$

(a)P\'agina Web

```
<blog>

 <owner>Pedro Ojeda</owner>

 <entry>
   <subject>culture</subject>
   <date>02-12-06</date>
   <content>blablabla1</content>
   <comments>1</comments>
 </entry>

 <entry>
   <subject>history</subject>
   <date>01-08-06</date>
   <content>blablabla2</content>
   <comments>0</comments>
 </entry>

</blog>
```

(b)T\'ermino del Algebra

```
blog(

owner(Pedro Ojeda),

  entry(
     subject(culture),
     date(02-12-6),
     content(blablabla1),
     comments(1)
  ),

  entry(
     subject(history),
     date(01-08-06),
     content(bla,bla,bla2),
     comments(0)
  )

)
```

Figura 3.1: Una página Web y su traducción a un término del álgebra

$\mathtt{C}$, con $Var(\mathtt{C}) \subseteq Var(\mathtt{l})$, donde $\mathtt{l}$ es un término, `error` es una constante reservada y $\mathtt{C}$ es una secuencia finita (posiblemente vacía) de ecuaciones (ej. $\mathtt{X} \in \mathtt{rexp}$) respecto a un lenguaje regular[1] y/o ecuaciones sobre los términos. Por un abuso de expresividad, también se permite escribir desigualdades de la forma $\mathtt{s} \neq \mathtt{t}$ en $\mathtt{C}$, la cual se cumple cuando la igualdad $\mathtt{s} = \mathtt{t}$ no es válida. Cuando $\mathtt{C}$ es vacío, simplemente se escribe $\mathtt{l} \rightharpoonup \mathtt{error}$. El significado de una regla de corrección $\mathtt{l} \rightharpoonup \mathtt{error} \,|\, \mathtt{C}$, donde $\mathtt{C} \equiv (\mathtt{X_1\ in\ rexp_1}, \ldots, \mathtt{X_n\ in\ rexp_n}, \mathtt{s_1} = \mathtt{t_1} \ldots \mathtt{s_m} = \mathtt{t_m})$, es el siguiente. Se dice que se cumple $\mathtt{C}$ para la substitución $\sigma$, si (i) cada estructura de texto $\mathtt{X_i}\sigma$, $\mathtt{i} = 1, \ldots, \mathtt{n}$, está contenida en el lenguaje de la correspondiente expresión regular $\mathtt{rexp_i}$; (ii) cada ecuación instanciada $(\mathtt{s_i} = \mathtt{t_i})\sigma$, $\mathtt{i} = 1, \ldots, \mathtt{m}$, se cumplen en $R$.

La página web $\mathtt{p}$ se considerada incorrecta si una instancia $\mathtt{l}\sigma$ de $\mathtt{l}$ es *recono-*

[1]El lenguaje regular se representa por medio de la usual sintaxis de expresiones regulares.

$p_1$) blog(owner(Pedro Ojeda),
entry(subject(culture),
date(02-12-06),
content(blablabla1),
comments(1)),
entry(subject(history),
date(01-08-06),
content(blablabla2),
comments(0)))

$p_2$) entry(subject(culture),
date(02-12-06),
comments(commentary(date(02-15-06),
author(blink(Pedro Ojeda)),
text(bla SEX bla))),
commentary(date(06-23-06),
author(Alan Turing),
text(blabla)))

$p_3$) entry(subject(history),
date(01-08-06),
comments())

$p_4$) members(member(name(Pedro Ojeda),age(28)),
member(name(Demis Ballis), age(32)),
member(name(Peter Pan), age(14)) )

Figura 3.2: Ejemplo de un sitio Web

*cida* dentro de p y C se cumple para $\sigma$.

El tercer conjunto de reglas $I_M$ permite especificar propiedades para detectar páginas Web incompletas o faltantes (*reglas de completitud*). Una regla de completitud se define como $\mathtt{l} \rightharpoonup \mathtt{r}\ \langle \mathtt{q} \rangle$, donde l y r son términos y $\mathtt{q} \in \{\mathtt{E}, \mathtt{A}\}$. Las reglas de completitud de una especificación Web formalizan el requerimiento de que alguna información deba ser incluida en todas o algunas de las páginas del sitio Web. Se usan los atributos $\langle \mathtt{A} \rangle$ y $\langle \mathtt{E} \rangle$ para distinguir las reglas "universales" de las "existenciales". La parte derecha de las reglas de completitud pueden contener funciones, que deberán estar definidas en $R$. Intuitivamente, la interpretación de

$r_1)$ $blink(X) \rightharpoonup error$
$r_2)$ $text(X) \rightharpoonup error|Xin[: TextTag :] * sex[: TextTag :]*$
$r_3)$ $member(age(X)) \rightharpoonup error|X < 18$
$r_4)$ $text(link(X)) \rightharpoonup error$
$r_5)$ $entry(commentary(author(X))) \rightharpoonup \sharp members(member(name(X)))\langle A\rangle$
$r_6)$ $blog(owner(X)) \rightharpoonup hpage(X)\langle E\rangle$

Figura 3.3: Ejemplo de una especificación Web

una regla universal $\texttt{l} \rightharpoonup \texttt{r}\ \langle\texttt{A}\rangle$ (respectivamente, una regla existencial $\texttt{l} \rightharpoonup \texttt{r}\ \langle\texttt{E}\rangle$) respecto a un sitio Web $W$ es el siguiente: si (una instancia de) `l` se reconoce en $W$, también (una instancia de) la forma irreducible de `r` debe ser reconocida en *todas* las (respectivamente, *algunas*) páginas Web las cuales embeban (una instancia de) `r`.

Otra posibilidad que permite el lenguaje es verificar las propiedades de completitud en un subconjunto de páginas del sitio Web. Para este propósito, se pueden marcar algunos símbolos de la parte derecha de la regla por medio del símbolo $\sharp$. La información marcada de la regla $r$ se utiliza para seleccionar el subconjunto de páginas Web del sitio sobre el cual vamos a verificar la propiedad. Más específicamente, la regla $r$ se ejecuta sobres las páginas que embeban la información marcada.

**Ejemplo 3.2.1** *Consideremos el sitio Web de la Figura 3.2 y la especificación Web de la Figura 3.3. Las reglas $r_1$, $r_2$, $r_3$ y $r_4$ son reglas de corrección, mientras que $r_5$ y $r_6$ son reglas de completitud. La regla $r_1$ representa la prohibición del uso de texto destellante dentro del sitio web. La regla $r_2$ limita la posibilidad de incluir comentarios con la palabra* sex *dentro del blog. La regla $r_3$ establece que la edad mínima para ser miembro del blog es de 18 años. La regla $r_4$ establece que no se pueden incluir* links *dentro de los textos. La regla $r_5$ especifica que el autor de un comentario debe pertenecer a los miembros del blog. La información marcada establece que la propiedad debe ser verificada solamente en las páginas donde esté la información de los miembros. La regla $r_6$ define que el propietario del blog debe tener una* home page. *La información de marcado en esta regla establece que la propiedad se debe verificar sólo por lo menos en una (por el uso del cuantificador existencial $\langle E\rangle$) de las páginas donde está la información de los*

*miembros del blog (páginas que contengan la palabra "*members*").*

La detección de errores se lleva a cabo por la ejecución de la especificación Web sobre el sitio Web. Esto se mecaniza por medio de *reescritura parcial*, un nueva técnica de reescritura que se obtiene remplazando el *pattern-matching* tradicional de la reescritura de términos con un mecanismo basado en *simulación de páginas* (árboles) (ver [Alpuente et al., 2006a]).

## 3.3. Simulación y reescritura parcial

La reescritura parcial extrae "algunas piezas de información" contenida en una página, y luego junta las piezas para reescribirlas a un nuevo término. La formalización se basa en simulación de árboles, los cuales reconocen la estructura y el etiquetado de un término dentro de una página particular del sitio Web.

La noción de simulación, $\trianglelefteq$, es una adaptación de la relación de *embedding* de Kruskal [Bezem, 2003], en donde ignoramos la regla de *diving*[2] [Leuschel, 2002].

**Definición 3.3.1 (simulación)** *La relación de* simulación

$$\trianglelefteq \subseteq \tau(\mathcal{T}ext \cup \mathcal{T}ag) \times \tau(\mathcal{T}ext \cup \mathcal{T}ag)$$

*sobre páginas Web es la relación que satisface la regla:*

$$\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_m) \trianglelefteq \mathtt{g}(\mathtt{s}_1, \ldots, \mathtt{s}_n) \textit{ sii } \mathtt{f} \equiv \mathtt{g} \textit{ y } \mathtt{t}_i \trianglelefteq \mathtt{s}_{\pi(i)}, \textit{ para } \mathtt{i} = 1, \ldots, \mathtt{m}, \textit{ y alguna funci on inyectiva } \pi : \{1, \ldots, \mathtt{m}\} \to \{1, \ldots, \mathtt{n}\}.$$

Dadas dos páginas Web $\mathtt{s}_1$ y $\mathtt{s}_2$, si $\mathtt{s}_1 \trianglelefteq \mathtt{s}_2$ se dice que $\mathtt{s}_1$ *simula* (o *es embebida* o *reconocida* dentro de) $\mathtt{s}_2$. También se dice que $\mathtt{s}_2$ *embebe* $\mathtt{s}_1$. Note que, en la Definición 3.3.1, para el caso donde $\mathtt{m}$ es 0 se tiene $\mathtt{c} \trianglelefteq \mathtt{c}$ para cada símbolo constante $\mathtt{c}$. Note también que $\mathtt{s}_1 \not\trianglelefteq \mathtt{s}_2$ si $\mathtt{s}_1$ o $\mathtt{s}_2$ contiene variables.

[2]La regla de *diving* permite "saltar" parte del término del lado derecho de la relación $\trianglelefteq$. Formalmente, $s \trianglelefteq f(t_1, \ldots t_n)$, si $s \trianglelefteq t_i$, para algún $i$.

A continuación se dará la noción de reescritura parcial, ignorando de momento las condiciones condiciones y/o cuantificadores de las reglas de la especificación Web.

**Definición 3.3.2 (reescritura parcial)** *Sean* $\mathtt{s}, \mathtt{t} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. *Entonces,* $\mathtt{s}$ reescribe parcialmente *a* $\mathtt{t}$ *via la regla* $\mathtt{l} \rightharpoonup \mathtt{r}$ *y la substitución* $\sigma$ *sii existe una posición* $u \in O_{\mathcal{T}ag}(\mathtt{s})$ *tal que*

1. $\mathtt{l}\sigma \trianglelefteq \mathtt{s}_{|u}$ *y,*
2. $\mathtt{t}$ = *Reduce*($\mathtt{r}\sigma$,*R), donde la función Reduce*$(x, R)$ *computa, por medio de reescritura estándar, la forma irreducible de* $x$ *en* $R$.

En línea general la Definición 3.3.2 dice que, dada una regla de especificación Web $\mathtt{l} \rightharpoonup \mathtt{r}$, la reescritura parcial permite extraer a partir de una página Web $\mathtt{s}$, una subparte de $\mathtt{s}$ la cual se simula por una instancia *ground* de $\mathtt{l}$, y remplazar $\mathtt{s}$ por una instancia *ground* reducida de $\mathtt{r}$. Note que el contexto de la expresión reducida seleccionada $\mathtt{s}_{|u}$ se desecha después de cada paso de reescritura. La notación $\mathtt{s} \rightharpoonup_I \mathtt{t}$, denota que $\mathtt{s}$ se reescribe parcialmente a $\mathtt{t}$ usando alguna regla que pertenece al conjunto $I$.

## 3.4. Verificación

Como vimos en la Sección 3.3, la simulación permite identificar la estructura de una página Web (o una *Web page template*) dentro de otra. Tomando en cuenta esto, la metodología propuesta en [Alpuente et al., 2006a] aplica simulación y reescritura parcial para verificar un sitio Web con respecto a una especificación Web y, de esta manera, detectar errores de corrección como así también de completitud dentro del sitio. Mas precisamente:

- *Errores de corrección*: información errónea o prohibida en las páginas Web.
- *Errores de completitud*: páginas Web faltantes en el sitio o páginas Web que son incompletas con respecto a una especificación Web.

A continuación se describe cómo podemos detectar estos errores de corrección y completitud.

### 3.4.1. Detección de errores de corrección

La metodología desarrollada en esta sección aplica las reglas de corrección a las páginas Web de un sitio Web para detectar patrones incorrectos.

Más precisamente, dada una página Web `p` y una regla de corrección `l` $\rightharpoonup$ `error` | `C`, primero se intenta reconocer `l` dentro `p` por medio de la reescritura parcial. Luego se analizan los valores tomados por las variables de `C`, los cuales se calculan en el paso de reescritura parcial. Si el texto estructurado, el cual está limitado a cada variables de `C`, pertenece al lenguaje de la correspondiente expresión regular y se cumplen todas las ecuaciones en `C` instanciadas, el fallo contenido dentro de la página Web `p` y la información del error detectado son entregados al usuario. Esta metodología permite precisar cuál es la posición y la información que el usuario debe modificar para poder reparar el error.

**Definición 3.4.1 (error de corrección)** *Sea $W$ un sitio Web y $(I_M, I_N, R)$ una especificación Web. Entonces, la cuádrupla $(\mathtt{p}, w, \mathtt{l}, \sigma)$ es un* error de corrección *sii $\mathtt{p} \in W$, $w \in O_{\mathcal{T}ag}(\mathtt{p})$ y $\mathtt{l}\sigma$ es una instancia de la parte izquierda `l` de una regla de corrección que pertenece a $I_N$ tal que $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$.*

Dado un error de corrección $e = (\mathtt{p}, w, \mathtt{l}, \sigma)$, `p` es la página donde se produce el error, $w$ es la posición del error dentro de la página, `l` es la parte izquierda de la regla que produjo el error y $\mathtt{l}\sigma$ representa la información errónea dentro de la página.

Se denota con $E_N(W)$ al conjunto de todos los errores de corrección de un sitio Web $W$ respecto del conjunto de reglas de corrección $I_N$. Cuando no exista confusión, simplemente se escribirá $E_N$.

**Ejemplo 3.4.2** *Consideremos el sitio Web de la Figura 3.2, y la especificación de la Figura 3.3. La metodología propuesta detecta los siguientes errores de corrección:*

- *La página $p_2$ del sitio contiene la sentencia blink, la cual está prohibida por la regla de corrección $r_1$ de la especificación*
  $e_{N1} = (p_2, 3.1.2.1, blink(X), \{X/Pedro\ Ojeda\})$.

- *La página $p_2$ del sitio, en el texto del comentario tenemos bla$SEX$bla, y mirando la regla $r_2$ tenemos que la palabra $SEX$ no está permitida dentro de los mismos*
  $e_{N2} = (p_2, 3.1.3, text(X), \{X/bla\ SEX\ bla\})$.

- *La página $p_4$ dice que $PeterPan$ tiene* 14 *años y, por la regla $r_3$, tenemos que la edad mínima es de* 18 *años*
  $e_{N3} = (p_4, 3.2, member(age(X)), \{X/14\})$.

*De esta manera tenemos que*
$E_N = \{e_{N1}, e_{N2}, e_{N3}\}$

### 3.4.2. Detección de errores de completitud

Esencialmente, la idea de detectar errores de completitud es calcular el conjunto de todas las posibles expresiones marcadas que puedan ser derivadas a partir del sitio Web $W$ vía las reglas de completitud de una especificación Web por medio de reescritura parcial. Estos términos marcados pueden verse como requerimientos a ser satisfechos por $W$. Entonces, se realiza la comprobación de errores si los requerimientos son satisfechos por $W$ usando simulación, la información del marcado y los cuantificadores. Resumiendo, el método trabaja en dos pasos:

1. Calcular el conjunto de requerimientos $\mathtt{Req}_{\mathtt{M,W}}$ para $W$ respecto a la $I_{\mathtt{M}}$;
2. Comprobar $\mathtt{Req}_{\mathtt{M,W}}$ en $W$.

Formalmente, un *requerimiento* es un par $\langle \mu(\mathtt{e}), \mathtt{q} \rangle$, donde $\mu(\mathtt{e})$ es un término marcado y $\mathtt{q} \in \{\mathtt{A}, \mathtt{E}\}$. Un requerimiento se dice *universal* si $\mathtt{q} = \mathtt{A}$, mientras que es *existencial* si $\mathtt{q} = \mathtt{E}$.

Se puede distinguir tres tipos de errores de completitud: (i) páginas Web faltantes, (ii) errores de completitud universal, y (iii) errores de completitud existencial. Estos tres tipos de errores $(M, A, E)$ pueden ser detectados por reescritura parcial sobre las páginas Web y la especificación $I_M$.

Más formalmente, tenemos la siguiente definición.

**Definición 3.4.3 (error de completitud)** *Sea $W$ un sitio Web, $(I_N, I_M, R)$ una especificación Web, y $c \in \{M, A, E\}$. Sea $q \in \{A, E\}$. Un error de completitud en $W$ es la tupla $e \equiv (s_0 \rightharpoonup^*_{I_M} s_{n-1} \rightharpoonup_{I_M} s_n, P, c)$ que satisface:*

*i)* *Existe una sustitución $\sigma$ s.t. $\mathtt{r}\sigma = s_n$ para alguna $\mathtt{l} \rightharpoonup \mathtt{r}\langle\mathtt{q}\rangle \in I_M$ .*
*ii)* *Existe una sustitución $\sigma'$ s.t. $\mathtt{l}'\sigma' = s_0$ para alguna $\mathtt{l}' \rightharpoonup \mathtt{r}'\langle\mathtt{q}'\rangle \in I_M$ .*
*iii)* *Existe $p \in W$ s.t. $s_0 \trianglelefteq p$ .*
*iv)* *$P = \{p \mid p \in mark(\mathtt{r}, W)$ y $s_n \not\trianglelefteq p\}$ .*
*v)* *$c = (M$ si $P = \emptyset)$ o $(\mathtt{q}$ si $P \neq \emptyset)$ .*

En la definición 5.5.1, $s_0 \rightharpoonup^*_{I_M} s_{n-1} \rightharpoonup_{I_M} s_n$ representa la secuencia de pasos de reescritura hasta detectar el término $s_n$ que no satisface la regla (por cuestión de simplicidad escribiremos $s_0 \rightharpoonup \ldots \rightharpoonup s_n$). Denotamos por $P$ el conjunto de páginas Web incompletas, mientras que $c$ es la clase o tipo del error detectado ($M$ *missing page*, $A$ *universal error*, y $E$ *existential error*). Note que, para un error de tipo *Missing Page*, el conjunto $P$ es vacío.

Denotamos con $\mathbb{E}_M(W)$ (o simplemente $\mathbb{E}_M$), al conjunto de todos los errores de completitud detectados en un sitio Web $W$. Cuando $|\mathbb{E}_M(W)| = 0$ decimos que $W$ está libre de errores de completitud o simplemente $W$ está reparado.

**Ejemplo 3.4.4** *Consideremos el sitio Web $W$ de la Figura 3.2, y la especificación $E$ de la Figura 3.3. Nuestra técnica detecta los siguientes errores de completitud son detectados:*

- *La página p2 del sitio contiene un comentario de* Alan Turing*, el cual no es miembro de este blog (ver los miembros en la página p4 del sitio), y por la regla de completitud $r_5$ obtenemos el* error de completitud existencial
  $e_{M1} \equiv (lhs_{r_5}\sigma \rightharpoonup_{r_5} rhs_{r_5}\sigma, \{p4\}, E)$, *donde* $\sigma = \{X/$*'Alan Turing'*$\}$.

- *Por la regla de completitud $r_6$, debería existir una página que sea la* home page *del propietario del blog, por lo tanto, tenemos el* error de página Web faltante
  $e_{M2} \equiv (lhs_{r_6}\sigma \rightharpoonup_{r_6} rhs_{r_6}\sigma, \emptyset, M)$, *donde* $\sigma = \{X/$*'Pedro Ojeda'*$\}$.

*De esta manera el conjunto de errores de completitud de $W$ respecto de $E$ es* $E_M = \{e_{M1}, e_{M2}\}$.

# 4

# Un motor algebráico para la verificación de sistemas web en GVerdi

Con el objetivo de mejorar el tiempo de respuesta de la herramienta *GVerdi* presentada en [Ballis, 2005; Ballis and Vivó, 2005] y de evaluar el rendimiento del lenguaje *Maude* a la hora de realizar la simulación de páginas web, describimos el nuevo motor para el sistema de verificación GVerdi desarrollado en Maude así como la extensión del sistema GVerdi para poder interactuar con el nuevo motor, que aporta mejoras significativas.

La metodología, como vimos en el Capítulo 3, está basada en una novedosa técnica de reescritura, llamada reescritura parcial, cuya implementación en Maude resulta optimizada gracias al pattern matching AC que soporta el lenguaje.

En este capítulo, en primer lugar, presentamos el lenguaje de programación Maude. A continuación, recordamos las principales características del lenguaje Maude que fueron explotadas por la implementación de nuestro motor de verificación de sitios Web. Luego, describimos el sistema de verificación GVerdi-M y por últimos mostramos las comparativas realizadas con GVerdi.

## 4.1. Maude

Los lenguajes de programación convencionales, también llamados imperativos, evolucionaron para explotar la arquitectura hardware de los ordenadores en los que eran ejecutados. Desde los años 70 hasta la actualidad, se ha constatado que

los lenguajes de programación deben evolucionar y liberarse de las restricciones impuestas por los computadores. En este sentido, los lenguajes de programación conocidos como declarativos poseen, en principio, las características adecuadas para dar este paso en la evolución.

Si bien el mayor nivel de abstracción de los programas declarativos proporciona toda una serie de ventajas, éste es también la causa de sus peores inconvenientes: (i) la complejidad y excasa facilidad de uso de los lenguajes de programación declarativos, y (ii) la dificultad para conseguir implementaciones eficientes.

Así, en los últimos años, la investigación en programación declarativa se ha centrado en mejorar las facilidades expresivas de los lenguajes de programación y en mejorar la eficiencia de sus implementaciones. Para ello, se han definido versiones concurrentes y orientadas a objetos de muchos lenguajes declarativos, se han desarrollado potentes entornos de programación que incluyen herramientas para asistir al programador (depuradores, analizadores, etc.) y se han implementado numerosas librerías para la programación de interfaces visuales, para realizar complejos cálculos matemáticos, para la programación distribuida, para analizar y generar código HTML, etc.

Sin embargo, la verdadera evolución reside en:

- La integración de los diferentes estilos de programación declarativa en un marco uniforme y más rico en capacidad expresiva, y
- En las capacidades de razonamiento automático sobre los programas especificados en dichos estilos.

Siguiendo estas pautas, surge la lógica de reescritura (o *rewriting logic*) desarrollada por José Meseguer. La teoría y aplicaciones de la lógica de reescritura han sido enérgicamente desarrolladas por investigadores de todo el mundo durante los últimos 12 años; principalmente USA, Francia y Japón. Durante estos años se han publicado más de 300 artículos en las revistas más prestigiosas, se han desarrollado varios lenguajes de programación denominados también declarativos (entre ellos Maude, ELAN, CafeOBJ, OBJ2 y OBJ3) y se han definido toda una amplia variedad de herramientas formales que han sido aplicadas satisfactoriamente por multitud de universidades y empresas de todo el mundo a un

amplio espectro de aplicaciones tales como modelos de computación, semánticas de lenguajes de programación, arquitecturas distribuidas, redes de Petri, actores y componentes software, demostración automática de teoremas, certificación de programas, verificación de protocolos de comunicación, etc.

Maude es el más completo de los lenguajes basados en la lógica de reescritura. La lógica ecuacional subyacente a la lógica de reescritura utilizada por Maude es la lógica ecuacional con pertenencia *(membership equational logic)* y da lugar a un modelo computacional más rico que permite definir teorías de reescritura generales y módulos orientados a objetos. El motor de reescritura de Maude es muy eficiente y tiene un alto rendimiento de ejecución gracias a un uso extensivo de avanzadas técnicas de semi-compilación; existe también un compilador eficiente de Maude aunque todavía experimental. Además, Maude proporciona reflexión de la lógica de reescritura de forma eficiente, una capacidad expresiva y computacional que facilita el razonamiento automático sobre los programas. Intuitivamente, una lógica es reflexiva si puede representar su propio meta-nivel al nivel objeto de una forma correcta y coherente. O en otras palabras, un programa Maude es capaz de manipular, modificar y ejecutar otros programas escritos en Maude de una forma tan eficiente como si se ejecutasen directamente en el propio lenguaje.

## 4.2. Verificación de sitios Web usando Maude

Maude es un lenguaje reflexivo, potente y versatil que soporta programación ecuacional y lógica de reescritura, lo cual es particularmente conveniente para el desarrollo de aplicaciones de dominio específico [Escobar et al., 2006a; Eker et al., 2003]. Maude no está solamente previsto para prototipo de sistemas, sino que se considera como un lenguaje de programación real con muy buenas prestaciones. A contiuación describimos las características del lenguaje Maude que fueron explotadas por la implementación de nuestro motor de verificación de sitios Web.

**Atributos ecuacionales.**

A continuación, describimos la manera en que modelamos (parte de) la rep-

```
fmod TREE-XML is
sort XMLNode .
op RTNode : -> XMLNode .                      -- Root information item
op ELNode _ _ : String AttList -> XMLNode . -- Element information item
op TXNode _ : String -> XMLNode .           -- Text information item
--- ...  definitions of the other XMLNode types omitted ...
sorts XMLTreeList XMLTreeSeq XMLTree .
op Tree (_) _ : XMLNode XMLTreeList - > XMLTree .
subsort XMLTree < XMLTreeSeq .
op _,_ : XMLTreeSeq XMLTreeSeq -> XMLTreeSeq    [comm assoc id:null] .
op null : -> XMLTreeSeq .
op [_] : XMLTreeSeq -> XMLTreeList .
op [] : -> XMLTreeList .
endfm
```

Figura 4.1: Ejemplo de codigo Maude

resentación interna de documentos XML en nuestro sistema. La representación modifica la estructura proporcionada por la librería Haskell HXML adicionando conmutatividad a la representación arbolea de XML. En otras palabras, en nuestro sistema, el orden de los hijos de los nodos en un árbol (término) no es relevante: e.g. $f(a, b)$ es "equivalente" a $f(b, a)$.

En el módulo de la Figura 4.1, el constructor `XMLTreeSeq _,_` tiene los atributos ecuacionales `comm assoc id:null`. Esto nos permite descartar los paréntesis de los nodos `XML` dentro de la lista. El significado de esta optimización se verá claramente cuando describamos el *AC pattern matching* y la forma en que lo usamos.

**AC pattern matching.**

El mecanismo de evaluación de Maude se basa en reescritura de términos módulo una teoría ecuacional $E$ (p.e. un conjunto de axiomas ecuacionales), la cual se logra por ejecutar un "ajuste de patrones" *(patter matching)* módulo la teoría ecuacional $E$. Más precisamente, dada una teoría ecuacional $E$ y los término $t$ y $u$, se dice que $t$ "empareja" con $u$ módulo $E$ (o que $t$ *E-matches* $u$), si existe una sustitución $\sigma$ tal que $\sigma(\mathrm{t})=_E u$, esto significa que, $\sigma(\mathrm{t})$ y $u$ son equivalentes módulo la teoría ecuacional $E$. Cuando $E$ contiene axiomas de asociatividad y conmutatividad para los operadores hablamos de *AC pattern matching*. El *AC*

*pattern matching* es un poderoso mecanismo de *matching* que nosotros empleamos para inspeccionar y extraer la estructura parcial de un término. En otras palabras, el *AC pattern matching* se utiliza directamente para la implementación de la noción del *homeomorphic embedding* de la Sección 3.3.

**Metaprogramación.**

Maude se basa en la lógica de reescritura [Martí-Oliet and Meseguer, 2002], la cual es reflexiva en el sentido matemático. En otras palabras, existe una teoría de reescritura finita $\mathcal{U}$ que es universal, es decir, se puede representar en $\mathcal{U}$ (como dato) cualquier teoría de reescritura finita $\mathcal{R}$ (incluyendo ella misma). De esta manera, se puede simular en $\mathcal{U}$ el comportamiento de $\mathcal{R}$. En nuestro sistema, utilizamos la capacidad de metaprogramación de Maude para implementar la semántica de las reglas de corrección y completitud. Esto es, durante el proceso de reescritura parcial, se crean y ejecutan módulos funcionales utilizando las características de metareducción del lenguaje.

Ahora estamos listos para explicar como implementamos la relación del *homeomorphic embedding*, dada en la Sección 3.3, utilizando las características mencionadas de alto nivel de Maude.

### 4.2.1. Implementación del *homeomorphic embedding.*

Consideremos dos plantillas de documentos XML $l$ y $p$. Los puntos críticos en nuestra metodología son (i) descubrir si $l \trianglelefteq p$ (e.g $l$ está embebido en $p$); (ii) en caso de que $l \trianglelefteq p$, encontrar la sustitución $\sigma$ tal que $l\sigma$ sea la instancia de $l$ reconocida dentro de $p$.

Podemos resumir nuestra propuesta como sigue: Utilizando las características de metanivel de Maude, primero construimos dinámicamente un módulo `M` que contiene la plantilla (regla) $l$ de la forma

$$\texttt{eq l} = \texttt{sub}(\texttt{"X}_1\texttt{"/X}_1), \ldots, \texttt{sub}(\texttt{"X}_n\texttt{"/X}_n), \qquad \texttt{X}_i \in \texttt{Var(l)}, \texttt{i} = 1, \ldots \texttt{n},$$

donde `sub` es un operador asociativo utilizado para registrar la sustitución $\sigma$ que buscamos computar. En el siguiente paso, intentamos reducir la plantilla $p$ utilizando la regla del módulo `M`, las plantillas $l$ y $p$ se representan internamente

por medio del constructor binario `_,_`, que tiene los atributos ecuacionales `comm assoc id:null` (ver la Sección 4.2). La ejecución del módulo `M` sobre $p$ esencialmente computa un *AC-matcher* entre $l$ y $p$. Es más, el *AC pattern matching* implementa directamente la relación del *homeomorphic embedding*. La ejecución del módulo `M` encuentra todas las subsumiciones homeomórficas de $l$ en $p$. Adicionalmente, por efecto de la ejecución de `M`, obtenemos las sustituciones $\sigma$ como una secuencia de variables instanciadas $X_i$, $\mathtt{i} = 1, \ldots, \mathtt{n}$

$$\mathtt{sub}(\text{"}\mathtt{X_1}\text{"}/\mathtt{X_1})\sigma, \ldots, \mathtt{sub}(\text{"}\mathtt{X_n}\text{"}/\mathtt{X_n})\sigma, \qquad \mathtt{X_i} \in \mathtt{Var(l)}, \mathtt{i} = 1, \ldots \mathtt{n},$$

**Ejemplo 4.2.1** *Sean $s_1$ y $s_2$ dos plantillas de documentos XML definidos como sigue*

$$s_1 \equiv hpage(surname(Y), status(prof.), name(X), teaching)$$

$$\begin{aligned} s_2 \equiv hpage(&name(mario), surname(rossi), status(prof.),\\ &teaching(course(logic1), course(logic2)),\\ &hobbies(hobby(reading), hobby(gardening))) \end{aligned}$$

*Construimos el módulo dinámico* `M` *conteniendo la regla*

$$\mathtt{eq\ hpage(surname(Y), status(prof), name(X), teaching)} = \mathtt{sub}(\text{"}\mathtt{Y}\text{"}/\mathtt{Y}), \mathtt{sub}(\text{"}\mathtt{X}\text{"}/\mathtt{X})\ .$$

*Dado que $s_1 \trianglelefteq s_2$, implica que existe un* AC-matcher *entre $s_1$ y $s_2$, el resultado de la ejecución de* `M` *sobre $s_2$ es la sustitución*

$$\mathtt{sub}(\text{"}\mathtt{Y}\text{"}/\mathtt{rossi}), \mathtt{sub}(\text{"}\mathtt{X}\text{"}/\mathtt{mario}).$$

## 4.3. El sistema de verificación GVerdi-M

Debido a que Maude no permite la realización de interfaces gráficas, y aprovechándonos de la versión previa de GVerdi desarrollada en Haskell en la que la parte de la interfaz estaba funcionalmente separada de la del motor, se ha reutilizado la parte de la interfaz gráfica para interactuar con el nuevo motor implementado en Maude. La forma de interactuar entre Haskell y Maude es mediante el uso de la invocación de un comando del sistema, tal y como veremos posteriormente. El sistema de verificación del que partíamos había sido desarrollado en Haskell

(GHC v6.2.2), estando disponible en http://www.dsic.upv.es/users/elp/GVerdi. La implementación del nuevo motor consta aproximadamente de 900 líneas de código.

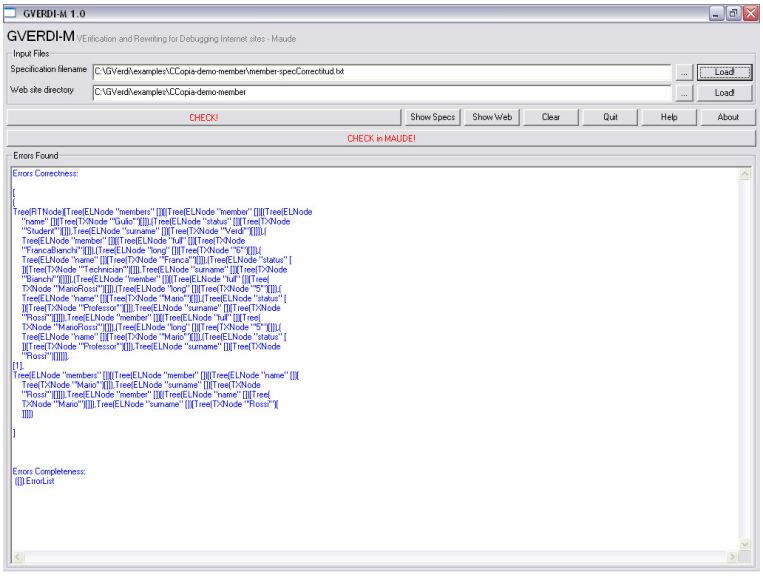


Figura 4.2: Sistema de verificación GVerdi-M

El sistema, incluye un analizador sintáctico para expresiones semiestructuradas (por ejemplo documentos XML/XMTL) y especificaciones web, módulos que implementan el mecanismo de reescritura parcial, la verificación y una interfaz gráfica tal y como se ve en la Figura 1. El sistema permite al usuario cargar un sitio web junto con una especificación web. Además, se pueden inspeccionar los datos cargados y comprobar el sitio web tanto utilizando la versión antigua del motor como la nueva. Se pueden detectar páginas web incompletas o perdidas y páginas web incorrectas de forma eficiente.

## 4.4. Comparativa

En los siguientes apartados utilizaremos distintos criterios para realizar distintas comparativas entre los motores implementados en Haskell y Maude.

### 4.4.1. Comparativa respecto a funcionalidad y tamaño

En primer lugar, recordemos que la implementación del motor de simulación en Haskell no es capaz de simular correctamente árboles con variables repetidas, mientras la versión de Maude sí es capaz, utilizando además un número menor de líneas de código, como veremos a continuación.

La tabla 4.1 resume la comparación entre las dos versiones:

| | **Motor en Haskell** | **Motor en Maude** |
|---|---|---|
| Líneas de código Simulación | 300 | 60 |
| Cuantificador Universal | No | Si |
| Cuantificador Existencial | No | Si |
| Reglas con variables repetidas | No | Si |

Cuadro 4.1: Funcionalidad GVerdi - GVerdi-M

Utilizando Maude, gracias al manejo de atributos de asociatividad y conmutatividad y las facilidades del metanivel, somos capaces de reducir considerablemente el número de líneas de la simulación con respecto a la implementación original de Gverdi. Para ser más precisos, pasamos de tener alrededor de 300 líneas de código en la implementación en Haskell a unas 60 líneas de código.

También, tenemos que destacar que en la anterior versión no se permitían la definición de los cuantificadores existenciales y/o universales en las reglas de completitud.

En resumen, con una menor cantidad de líneas de código, obtuvimos una mayor funcionalidad.

### 4.4.2. Comparativa respecto al tiempo de ejecución

Ahora presentamos una comparativa entre los motores en base al tiempo de ejecución. Para este estudio se ha utilizado un ordenador con las siguientes características:

Procesador: mobile AMD Athlon 2000+
RAM: 768 Mb

Los datos de pruebas, para la realización de esta comparativa han sido:

- Para la definición del Sitio Web, se ha elegido como base la página web ($p_4$) del sitio web mostrado en la Figura 3.2. A partir de esta página web, se ha ido modificando el número de miembros, lo que define el tamaño de la carga del sitio web.

- La regla $r$ utilizada para comprobar la página ha sido:
  $r = member(name(X), surname(Y)) - >$
  $\sharp member(name(X), surname(Y), full(appendXY))$

En la tabla 4.2mostramos los resultados en la siguiente tabla:

| **Número Miembros** | **Motor en Haskell** | **Motor en Maude** |
|---:|---:|---:|
| 10 | $30,834$ s | $2,01$ s |
| 50 | 25 m 18 s | $3,234$ s |
| 100 | $\infty$ | $4,375$ s |
| 300 | $\infty$ | $10,886$ s |
| 500 | $\infty$ | $28,132$ s |
| 1000 | $\infty$ | 1 m $29,234$ s |

Cuadro 4.2: Tiempos de ejecución GVerdi - GVerdi-M

En resumen, de acuerdo con todo lo anterior, vemos que el nuevo motor procesa de manera más versátil y en una cantidad muy inferior de tiempo la verificación de un sitio web.

# 5

# Reparación de errores

Con el objetivo de reparar un sitio Web que contiene errores, en [Alpuente et al., 2006b] se definen tres acciones reparadoras básicas de corrección que permiten eliminar la inconsistencia y/o añadir la información ausente. Las acciones consideradas son las siguientes:

- **delete**$(p, t)$ elimina todas las ocurrencias del término $t$ en de la página Web $p$, y devuelve la página Web modificada.
- **insert**$(p, w, t)$ modifica la página Web $p$ añadiendo el término t en $p_{|w}$.
- **add**$(p, W)$ añade la página Web $p$ al sitio Web $W$.

Note que las acciones reparadoras *add* e *insert* introducen nueva información en el sitio Web que puede contener/provocar nuevos errores. Por ello, es importante restringir la información que se puede añadir. Del mismo modo, la ejecución de la acción *delete* puede inducir nuevos errores de completitud. Para prevenir estas situaciones, en [Alpuente et al., 2006b] formalizamos las propiedades *safe* y *acceptable* que garantizan el comportamiento seguro de estas acciones.

A groso modo, estas propiedades nos aseguran, respectivamente, que no se introducen nuevos errores, y que la cantidad de errores en el sitio Web decrece tras la ejecución de cada acción reparadora. Estas propiedades garantizan la terminación del proceso de corrección.

A lo largo de este trabajo, asuminos que la aplicación de las acciones reparadoras es siempre *safe* y *acceptable*.

A continuación, describiremos como reparar un error de corrección/completitud. Daremos más detalles sobre los errores de completiud, ya que es en esta parte donde he participado más activamente.

## 5.1. Reparación de errores de corrección

El procedimiento de reparar los errores de corrección es el siguiente: siempre que se detecta un error de corrección, se elije una posible acción reparadora (según las diferentes acciones descritas), que será ejecutada para eliminar la información errónea, siempre que la aplicación de la acción sea *acceptable*.

Dado un error $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$, $\mathtt{e}$ puede ser reparado de dos maneras diferentes:

i) Eliminado el contenido incorrecto $\mathtt{l}\sigma$ de la página Web $\mathtt{p}$ (específicamente, $\mathtt{p}_{|w}$); para ello, es suficiente aplicar la acción reparadora $\mathbf{delete}(\mathtt{p}, \mathtt{p}_{|w})$[1].

ii) Cambiando $\mathtt{l}\sigma$ por un término que sea correcto; el proceso remplaza el subtérmino $\mathtt{p}_{|w}$ de la página Web $\mathtt{p}$ con un nuevo término $t$ introducido por el usuario[2].

## 5.2. Reparación de errores de completitud

En esta sección se formulan las operaciones para reparar un sitio Web que contenga errores de completitud. Sin pérdida de generalidad, se asume que el sitio Web $W$ es incompleto pero correcto con respecto a una especificación Web $(I_N, I_M, R)$. Esto se justifica por la posibilidad de asumir el haber ejecutado previamente la metodología de reparación de la corrección de [Alpuente et al., 2006b]. Tal suposición permite diseñar una metodología reparadora la cual "completa" el sitio Web sin introducir información incorrecta. Como dijimos en la sección anterior, cualquier error $e \in \mathbb{E}_M(W)$ puede ser reparado añadiendo la información ausente o eliminando el dato que lo produjo. De igual forma, consideramos *acceptable* a cualquier acción que se ejecute, lo cual se puede comprobar fácilmente ejecutando la metodología de verificación de [Alpuente et al., 2006a].

[1] Note que, en lugar de eliminar todo el subtérmino $\mathtt{p}_{|w}$, otra opción más precisa, pero también con un coste mayor, podría ser una implementación de la acción **delete** que elimine sólo las partes de $\mathtt{l}\sigma$ de $\mathtt{p}_{|w}$ que son responsables de que ocurra el error de corrección.

[2] Note que no es suficiente comprobar que el nuevo término no contenga errores, también es necesario verificar que, al insertarlo en la página, no se genere un nuevo error.

A continuación describimos cómo es posible reparar los errores de completitud detectados en un sitio Web. En primer lugar, veremos cómo añadir la información necesaria; a continuación, describiremos cómo eliminar la información incompleta que provocó el error.

## 5.3. Añadiendo información

Dependiendo del tipo de error de completitud, tenemos dos posibles acciones reparadoras a ser ejecutadas, $\mathbf{add}(\mathtt{p}, W)$ e $\mathbf{insert}(\mathtt{p}, w, t)$. La acción *add* se usa cuando tenemos un *missing page error* mientras la acción *insert* se ejecuta cuando el error de completitud es *universal error* o *existential error*. A continuación, analizaremos como reparar cada uno de ellos.

**Reparar un *Missing Page error.*** Sea $e \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P, q)$ un error de página ausente, donde $P = W$ y $q = M$. Para reparar $e$ añadiremos al sitio Web $W$ una página Web $\mathtt{p}$ que subsuma o embeba la expresión $\mathtt{s_n}$. De esta manera tenemos

$$W = W \cup \{\mathbf{add}(\mathtt{p}, W)\}, \text{ donde } \mathtt{s_n} \trianglelefteq \mathtt{p}_{|w} \text{ para algún } w \in O_{\mathcal{T}ag}(\mathtt{p}). \tag{5.1}$$

**Reparar un *Existential error.*.** Sea $e \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P, q)$ un error existencial, donde $q \equiv E$. Reparamos el error insertando el término $s_n$ en una página arbitraria $\mathtt{p}$ tal que $\mathtt{p} \in P$. De esta manera tenemos

$$W = W \setminus \{\mathtt{p}\} \cup \{\mathbf{insert}(\mathtt{p}, w, s_n)\}, \text{ donde } \mathtt{s_n} \trianglelefteq \mathtt{p} \text{ para algún } w \in O_{\mathcal{T}ag}(\mathtt{p}). \tag{5.2}$$

**Reparar un *Universal error*** Sea $e \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P, q)$ un error universal, donde $q \equiv A$. Reparamos el error insertando el término $s_n$ en cada página Web $\mathtt{p} \in P$. De esta manera, el sitio Web $W$ se transforma como sigue

$$W = W \setminus \{\mathtt{p}\} \cup \{\mathbf{insert}(\mathtt{p}, w, s_n)\} \quad \forall \mathtt{p} \in P, \text{ y } \exists w \in O_{\mathcal{T}ag}(\mathtt{p}) \tag{5.3}$$

Note que la información añadida para reparar un error podría casualmente ser la misma que sirva para reparar otro error. Un análisis sobre la dependencia entre errores se da en la Sección 5.5.1.

## 5.4. Eliminando información incompleta

En algunas situaciones es más conveniente eliminar la información incompleta. En particular, esta opción puede resultar particularmente útil cuando se tiene información desactualizada. La principal idea es eliminar aquella información del sitio Web que causa el error de completitud. El tratamiento de la información incompleta es independiente del tipo de error $(M, A, E)$ que estemos analizando; por lo tanto, la información ausente se computa de la misma manera para los tres tipos de errores.

A continuación, damos la definición de la operación *repairByDelete*.

**Definición 5.4.1 (***repairByDelete***)** *Dado un sitio Web $W$ y un error de completitud $e \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P, q)$, el sitio Web $W$ se transforma aplicando acciones de borrado de la siguiente manera.*

$$\begin{aligned} repairByDelete(e, W) = \; & \{\mathtt{p} \in W \mid s_0 \not\trianglelefteq \mathtt{p}_{|w}, \forall\, w \in O_{\mathcal{T}ag}(\mathtt{p})\} \cup \\ & \{\mathbf{delete}(\mathtt{p}, s_0) \mid \mathtt{p} \in W, s_0 \trianglelefteq \mathtt{p}_{|w}, w \in O_{\mathcal{T}ag}(\mathtt{p})\} \end{aligned}$$

En otras palabras, en la Definición 5.4.1 se elimina de todas las páginas Web la ocurrencia del término $s_0$ que inicia la secuencia de reescritura. Note que, si en la secuencia de reescritura de dos errores tenemos el mismo término inicial, es posible que la reparación de un error corrija de manera automática a otro.

## 5.5. Estrategias de reparación para sitios Web incompletos

El lenguaje de especificación Web, junto con la técnica de verificación y validación de propiedades formales sobre sitios Web, permite detectar páginas Web incompletas o ausentes en un sitio Web, obteniendo así un conjunto de errores que denominamos "errores de completitud". Estos errores representan la información inconsistente o ausente en el sitio Web.

Como ya hemos visto, podemos distinguir tres tipos de errores de completitud: *Missing page* (o página ausente), cuando una expresión no aparece en el sitio Web; *Universal completenes error* (o error universal), *Existential completeness error* (o error existencial). Un error *universal* (resp. *existential*) corresponde

a la satisfacción de una condición *universal* (resp. *existential*) del lenguaje de especificación.

Como vimos en el Capítulo 3, estos tres tipos de errores $(M, A, E)$ pueden ser detectados por reescritura parcial sobre las páginas Web y la especificación $I_M$.

En esta sección consideraremos una extensión de los errores de completitud en donde añadiremos a la información del error la secuencia de pases de reescrituras que fueron necesarios realizar hasta detectar el término que no satisface la regla. Más formalmente tenemos, la siguiente definición.

**Definición 5.5.1 (error de completitud)** *Sea $W$ un sitio Web, $(I_N, I_M, R)$ una especificación Web, y $c \in \{M, A, E\}$. Sea $q \in \{A, E\}$. Un error de completitud en $W$ es la tupla $e \equiv (s_0 \rightharpoonup^*_{I_M} s_{n-1} \rightharpoonup_{I_M} s_n, P, c)$ que satisface:*

- *i)* *Existe una sustitución $\sigma$ s.t. $\mathtt{r}\sigma = s_n$ para alguna $\mathtt{l} \rightharpoonup \mathtt{r}\langle \mathtt{q} \rangle \in I_M$ .*
- *ii)* *Existe una sustitución $\sigma'$ s.t. $\mathtt{l}'\sigma' = s_0$ para alguna $\mathtt{l}' \rightharpoonup \mathtt{r}'\langle \mathtt{q}' \rangle \in I_M$ .*
- *iv)* *$P = \{p \mid p \in mark(\mathtt{r}, W)$ y $s_n \not\trianglelefteq p\}$ .*
- *v)* *$c = (M$ si $P = \emptyset)$ o $(\mathtt{q}$ si $P \neq \emptyset)$ .*

En la definición 5.5.1, $s_0 \rightharpoonup^*_{I_M} s_{n-1} \rightharpoonup_{I_M} s_n$ representa la secuencia de pasos de reescritura hasta detectar el término $s_n$ que no satisface la regla (por cuestión de simplicidad escribiremos $s_0 \rightharpoonup \ldots \rightharpoonup s_n$). Denotamos por $P$ el conjunto de páginas Web incompletas, mientras que $c$ es la clase o tipo del error detectado (*$M$ missing page*, *$A$ universal error*, y *$E$ existential error*). Note que, para un error de tipo *Missing Page*, el conjunto $P$ es vacío.

Denotamos con $\mathbb{E}_M(W)$ (o simplemente $\mathbb{E}_M$), al conjunto de todos los errores de completitud detectados en un sitio Web $W$. Cuando $|\mathbb{E}_M(W)| = 0$ decimos que $W$ está libre de errores de completitud o simplemente $W$ está reparado.

**Ejemplo 5.5.2** *Considere el sitio Web $W$ de la Figura 3.2, y la especificación $E$ de la Figura 3.3. Los siguientes errores de completitud son detectados:*

- *La página p2 del sitio contiene un comentario de* Alan Turing*, el cual no es miembro de este blog (ver los miembros en la página p4 del sitio), y por la regla de completitud $r_5$ tenemos el* error de completitud existencial *$e_{M1} \equiv (lhs_{r_5}\sigma \rightharpoonup_{r_5} rhs_{r_5}\sigma, \{p4\}, E)$, donde $\sigma = \{X/$'Alan Turing'$\}$.*

- *Por la regla de completitud $r_6$, debería existir una página que sea la* home page *del propietario del blog, por lo tanto, tenemos el* error de página Web faltante
  $e_{M2} \equiv (lhs_{r_6}\sigma \rightharpoonup_{r_6} rhs_{r_6}\sigma, \emptyset, M)$, *donde* $\sigma = \{X/\text{'Pedro Ojeda'}\}$.

*De esta manera el conjunto de errores de completitud de $W$ w.r.t. $E$ es* $E_M = \{e_{M1}, e_{M2}\}$.

### 5.5.1. Dependencia entre errores de completitud

Un sitio Web puede contener varios errores de completitud que podrían estar de alguna manera conectados. Por otro lado, con la ejecución de una operación reparadora es posible reparar más de un error. A continuación, analizamos cómo es la dependencia entre errores de completitud.

En primer lugar definimos dos órdenes parciales sobre el conjunto de errores $\mathbb{E}_M$, que están inducidos por los términos de la secuencia de reescritura parcial que llevan a la manifestación del error.

**Definición 5.5.3** *(orden inducido por los inferiores – $\preceq_{inf}$) Sean* $e_1 \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P_1, q_1)$ *y* $e_2 \equiv (t_0 \rightharpoonup \ldots \rightharpoonup t_m, P_2, q_2)$ *dos errores de completitud en* $\mathbb{E}_M(W)$. *Entonces,*

$$e_1 \preceq_{inf} e_2 \text{ sii } s_0 \trianglelefteq t_0.$$

*Diremos que un error* $e \in \mathbb{E}_M(W)$ *es* minimal *w.r.t.* $\preceq_{inf}$, *sii no existe* $e'$ *s.t.* $e' \preceq_{inf} e$ *y* $e' \not\equiv e$.

**Definición 5.5.4** *(orden inducido por los superiores – $\preceq^{sup}$) Sean* $e_1 \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P_1, q_1)$ *y* $e_2 \equiv (t_0 \rightharpoonup \ldots \rightharpoonup t_m, P_2, q_2)$ *dos errores de completitud en* $\mathbb{E}_M(W)$. *Entonces,*

$$e_1 \preceq^{sup} e_2 \text{ sii } s_n \trianglelefteq t_m.$$

En la Definición 5.5.3 se comparan los errores observando la relación de simulación que existe entre los términos que iniciaron la secuencia de reescritura de cada error. En la Definición 5.5.4, se observan en cambio los términos finales de esta secuencia. Decimos que $e_1$ y $e_2$ no son comparables w.r.t. $\preceq_{inf}$ (resp. $\preceq^{sup}$) sii $e_1 \not\preceq_{inf} e_2$ (resp. $e_1 \not\preceq^{sup} e_2$) y $e_2 \not\preceq_{inf} e_1$ (resp. $e_2 \not\preceq^{sup} e_1$).

Explotando estas dos definiciones sobre los errores obtenemos la siguiente proposición, que establece que reparar el menor de los errores ($e_1$) con respecto a la relación $\preceq_{inf}$, utilizando la operación *repairByDelete*, permite reparar de manera automática el resto de los errores que están relacionados con $e_1$ según el órden $\preceq_{inf}$.

**Proposición 5.5.5** *Sea $W$ un sitio Web, y sean los errores de completitud en el sitio Web $e_i \in \mathbb{E}_M(W), i = 1 \ldots m$, y sea $e_1 \preceq_{inf} \ldots \preceq_{inf} e_m$. Entonces, realizando la acción reparadora $repairByDelete(e_1, W)$ se reparan todos los errores $e_1, \ldots, e_m$ en $W$.*

**Proof.** Si se cumple la relación $e_1 \preceq_{inf} \ldots \preceq_{inf} e_m$, por la Definición 5.5.3 tenemos $s_{1_0} \trianglelefteq s_{2_0} \trianglelefteq \ldots \trianglelefteq s_{m_0}$. Por tanto, al ejecutar $repairByDelete(e_1, W)$ se elimina el término $s_{1_0}$ de $W$ (por la relación $\trianglelefteq$, también se elimina un subtérmino de $s_{2_0}$ y así sucesivamente hasta $s_{m_0}$). Junto con ellos se eliminan también los respectivos errores. □

Note que, la proposición 5.5.5 es independiente del tipo que sean los errores (*Missing page, Universal, o Existential error*).

Veamos ahora cómo reparar el sitio Web añadiendo la información ausente y siendo $e_1 \preceq^{sup} e_2$. ¿Es posible en esta situación reparar de manera automática más de un error?. A continuación, profundizaremos el análisis sobre la relación entre errores de completitud con el objetivo de responder a esta pregunta.

En primer lugar, veremos algunas consideraciones necesarias:

- El orden de errores de completitud $\preceq^{sup}$ permite conocer cuál es el error que, al ser reparado, añadirá más información al sitio Web.
- El orden para tratar los errores debe ir de mayor a menor w.r.t. $\preceq^{sup}$; de esta manera, cuando se repare un error se añade información útil para los errores menores.
- Si $e_n$ (es el mayor error w.r.t. $\preceq^{sup}$) es un error de *missing page* y/o existencial, de manera automática se reparan todos los errores de *missing page* y/o existenciales inferiores. Esto se debe a que, de una sola vez, se añade

información que embebe a la información ausente indicada por los demás errores. Note que reparar un error universal actuará de la misma manera sobre los errores *missing page* y/o existenciales inferiores.

- Cuando se repara un error universal, es posible que resulten reparadas otras páginas que pertenezcan a otro error universal inferior y, por lo dicho anteriormente, no pertenecen más al error.
- Una vez aplicada alguna acción reparadora en una página, no es necesario volver a aplicarle otra acción en un error inferior en el orden $\preceq^{sup}$.

Estas consideraciones, están expresadas en el Algoritmo 1, que describe el procedimiento $repairByInsert$ que permite reparar errores de completitud ordenados por la relación $\preceq^{sup}$.

Los resultados obtenidos en la Proposición 5.5.5 y el Algoritmo 1, permiten una optimización obvia del marco de reparación, que formalizamos en las estrategias de reparación presentadas a continuación.

### 5.5.2. Estrategias de reparación

Cómo explicamos en la Sección 5.5.1, es posible reparar un error de completitud por la ejecución de alguna acción reparadora. Por $(e, a)$ denotamos el par que contiene una acción reparadora $a$ que corrije el error $e$. Además, por la notación $W' = a(e, W)$ especificamos la ejecución de la acción reparadora $a$ sobre el sitio Web $W$, la cual retorna el sitio Web $W'$ con el error $e$ reparado.

Llamaremos *estrategia de reparación* a la ejecución de una secuencia de acciones reparadoras que permitan reparar todos los errores detectados en un sitio Web. Mas formalmente tenemos.

**Definición 5.5.6 (estrategia de reparación)** *Sea $W$ un sitio Web y sea $\mathbb{E}_M(W) = \{e_1, \ldots, e_n\}$ el conjunto de errores de completitud en $W$. Una* estrategia de reparación *para $W$ es la secuencia $[(e_1, a_1), \ldots, (e_n, a_n)]$, donde $a_1, \ldots, a_n$ son acciones reparadoras s.t.*

$$(i)\ W_0 = W;$$
$$(ii)\ W_i = a_i(e_i, W_{i-1})\ \forall i, 1 \leq i \leq n;$$

**Algorithm 1** Procedimiento para reparar errores de completitud ordenados por la relación $\preceq^{sup}$.

**Require:**
$\mathbb{E} = e_i \in \mathbb{E}_M(W), i = 1, \ldots m$, and $[e_1 \preceq^{sup} \ldots \preceq^{sup} e_m]$
Un sitio Web $W$
**Ensure:**
$W \mid \forall e \in \mathbb{E}, e \notin \mathbb{E}_M(W)$

```
1: procedure repairByInsert (𝔼, W)
2:   P_R = {} // conjunto de páginas reparadas
3:   for i ← m to 1 do
4:     (s_0 ⇀ ... ⇀ s_n, P, q) ← e_i
5:     if q = M and P_R = {} then
6:       W ← W ∪ {add(s_n, W)}
7:       P_R ← P_R ∪ {s_n}
8:     else if q = E and P_R = {} then
9:       p ← element(P) // obtener una página
10:      p' ← insert(p, w, s_n) // w es una posición arbitraria en p
11:      W ← W \ {p} ∪ {p'}
12:      P_R ← P_R ∪ {p}
13:    else if q = A then
14:      P_Aux ← P \ P_R
15:      for all p ∈ P_Aux do
16:        p' ← insert(p, w, s_n) // w es una posición arbitraria en p
17:        W ← W \ {p} ∪ {p'}
18:        P_R ← P_R ∪ {p}
19:      end for
20:    end if
21:  end for
22: end procedure
```

*y entonces,* $\mathbb{E}_M(W_n) = \emptyset$.

En la Sección 5.5.1 vimos cómo, al reparar un error de completitud, es posible reparar de manera automática otro error. Este hecho nos sugiere que no es necesario ejecutar una acción reparadora por cada error detectado en un sitio Web. A continuación, presentamos dos posibles estrategias de reparación. En la primera, el objetivo es reducir la cantidad de información a eliminar para obtener un sitio Web libre de errores; en la segunda, en cambio, se persigue reducir la cantidad de información que se debe añadir. En ambos casos, sólo es necesario reparar un subconjunto de los errores del sitio Web.

### A. Estrategia *reduce-delete-actions*

La relación $\preceq_{inf}$ define un orden parcial sobre el conjunto de errores de completitud $\mathbb{E}_M$. Por otro lado, en la Proposición 5.5.5, vimos que reparar un error *minimal* w.r.t. $\preceq_{inf}$ por medio de la operación $repairByDelete$ permite reparar los demás errores relacionados con él según este orden. Es claro ver que si tenemos dos errores *minimales* $e_1$ y $e_2$ no comparables w.r.t. $\preceq_{inf}$ es necesario reparar ambos errores.

**Definición 5.5.7 (estrategia *reduce-delete-actions* )** *Sea $W$ un sitio Web y sea $\mathbb{E}_M(W)$ el conjunto de errores de completitud de $W$. Una* estrategia de reparación *(o estrategia $\mathbb{RDA}$) que permite reducir las acciones de eliminación para $W$ es*

$$[(e_1, repairByDelete), \ldots, (e_n, repairByDelete)],$$
$$\textit{donde } \forall i, 1 \leq i \leq n, e_i \in \mathbb{E}_M(W) \textit{ y es minimal w.r.t. } \preceq_{inf}$$

La Definición 5.5.7 determina la estrategia $\mathbb{RDA}$, la cual consiste en reparar todos los errores minimales con respecto a la relación $\preceq_{inf}$ de un sitio Web. Esto nos lleva a la siguiente proposición, que establece que la estrategia $\mathbb{RDA}$ permite obtener un sitio Web $W$ libre de errores reparando sólo un subconjunto de los errores de $\mathbb{E}_M(W)$.

**Proposición 5.5.8** *Sea $W$ un sitio Web y sea $\mathbb{E}_M(W)$ el conjunto de errores de completitud de $W$. Entonces, $(i)$ la estrategia $\mathbb{RDA}$ obtiene un sitio Web libre de errores; $(ii)$ la cantidad de acciones reparadoras ejecutadas por la estrategia $\mathbb{RDA}$ es menor o igual al número original de errores en $\mathbb{E}_M$.*

**Proof.** Sea $e \in \mathbb{E}_M(W)$, dos situaciones son posibles: $(i)$ si $e$ es minimal w.r.t. $\preceq_{inf}$, $e$ es reparado con la operación $repairByDelete$; $(ii)$ si $e$ no es minimal w.r.t. $\preceq_{inf}$, entonces existe un minimal $e' \in \mathbb{E}_M$ s.t. $e' \preceq_{inf} e$ que será reparado y, por la Proposición 5.5.5, $e$ será reparado de manera automática sin necesidad de ejecutar una acción reparadora para $e$. □

### B. Estrategia *reduce-insertion-actions*

El procedimiento *repairByInsert* (descrito en el Algoritmo 1), nos permite reducir la cantidad de información que se debe añadir, como así también la cantidad de acciones reparadoras de inserción a ser ejecutadas en un sitio Web.

Ahora observemos la siguiente situación, donde un error pertenece a más de un conjunto de errores definidos por la relación $\preceq^{sup}$. Sean $\alpha = \{e_i\}_{i=1}^{n}$ y $\beta = \{e'_j\}_{j=1}^{m}$ dos subconjuntos de errores de $\mathbb{E}_M(W)$, tal que $e_1 \preceq^{sup} \ldots \preceq^{sup} e_n$ y $e'_1 \preceq^{sup} \ldots \preceq^{sup} e'_m$, y sea sea $e$ un error de completitud s.t. $e \in \alpha$ y $e \in \beta$. Es claro ver que un error puede pertenecer a más de un conjunto de errores definidos por la relación $\preceq^{sup}$.

Llamaremos $\mathsf{C}_{\mathbb{E}_M}$ a la secuencia de conjuntos de errores formados por la relación $\preceq^{sup}$ como sigue

$$\begin{array}{rl} \mathsf{C}_{\mathbb{E}_M}(\mathbb{E}_M, \preceq^{sup}) = & [c_1, \ldots, c_n] \\ \text{s.t.} & (\forall e \in \mathbb{E}_M,\ \exists i, 1 \leq i \leq n,\ \text{s.t. } e \in c_i), \\ & (\forall i, 1 \leq i \leq n,\ \forall e_1, e_2 \in c_i,\ e_1 \preceq^{sup} e_2 \text{ o } e_2 \preceq^{sup} e_1) \text{ y} \\ & (\forall i, 1 \leq i \leq n,\ |c_i| \geqslant |c_{i+1}|) \end{array}$$

La secuencia $\mathsf{C}_{\mathbb{E}_M}$ está ordenada por la cardinalidad de los conjuntos que la componen. Denotaremos por $\mathsf{C}_{\mathbb{E}_M}(i)$ a la secuencia $[c_1, \ldots, c_i]$.

De esta manera, podemos definir una partición sobre el conjunto $\mathbb{E}_M$ de la siguiente forma

$$\begin{array}{rl} \Gamma(\mathbb{E}_M) = \{m_i \mid m_i = & dif(\mathsf{C}_{\mathbb{E}_M}(i)),\ \forall i, 1 \leq i \leq k,\ k = |\mathsf{C}_{\mathbb{E}_M}|\}, \\ \text{donde} & dif([x_0]) = x_0 \\ & dif([x_0, \ldots, x_n]) = x_n \setminus \ldots \setminus x_0,\ \text{si } n > 0 \end{array}$$

**Definición 5.5.9 (estrategia *reduce-insertion-actions*)** *Sea $W$ un sitio Web y sea $\mathbb{E}_M(W)$ el conjunto de errores de completitud de $W$. Una estrategia para reducir la información a insertar (o estrategia $\mathbb{RIA}$) para $W$ es*

$$\begin{array}{c} [(m_1, repairByInsert), \ldots, (m_n, repairByInsert)], \\ \textit{donde } \forall i, 1 \leq i \leq n,\ m_i \in \Gamma(\mathbb{E}_M(W)) \end{array}$$

La estrategia de la Definición 5.5.9 ejecuta el procedimiento *repairByInsert* en cada conjunto de la partición de $\mathbb{E}_M$.

**Proposición 5.5.10** *Sea $W$ un sitio Web y sea $\mathbb{E}_M(W)$ el conjunto de errores de completitud de $W$ y sea $\Gamma(\mathbb{E}_M)$ una partición sobre $\mathbb{E}_M$. Entonces, (i) la estrategia $\mathbb{RIA}$ obtiene un sitio Web libre de errores; (ii) la cantidad de acciones reparadoras ejecutadas por la estrategia $\mathbb{RIA}$ es menor o igual al número original de errores en $\mathbb{E}_M$.*

**Proof.** Los conjuntos de la partición $\Gamma(\mathbb{E}_M)$ están ordenados w.r.t. $\preceq^{sup}$ y, como vimos en el Algoritmo 1, la ejecución del procedimiento $repairByInsert$ reduce la cantidad de información necesaria a añadir. De esta manera, con la ejecución de la estrategia $\mathbb{RIA}$ reparamos todos los errores de completitud del sitio Web. □

A continuación se presenta un ejemplo para clarificar la utilización de las estrategias *reduce-delete-actions* y *reduce-insertion-actions*.

**Ejemplo 5.5.11** *Sea el sitio Web $W$ formado por el conjunto de páginas $\{p_1, p_2, p_3, p_4\}$; y la especificación Web $(I_N, I_M, R)$ con $I_M = \{r_1, r_2, r_3, r_4\}$, de la siguiente forma*

| *Sitio Web* $W = \{p_1, p_2, p_3, p_4\}$ | *Especifcación Web* $(I_M, I_N, R)$ |
|---|---|
| $p_1 = m(s(b), f(a))$ | $r_1 = f(X) \rightharpoonup \sharp g(X)\langle A\rangle$ |
| $p_2 = m(m(g(a)))$ | $r_2 = g(X) \rightharpoonup \sharp h(X)\langle E\rangle$ |
| $p_3 = m(l(b, a))$ | $r_3 = h(X) \rightharpoonup \sharp p(X)\langle A\rangle$ |
| $p_4 = h(b)$ | $r_4 = l(X, Y) \rightharpoonup \sharp p(X, Y)\langle A\rangle$ |

*El conjunto de errores de completitud $E_M$ detectados por el proceso de verificación es: $E_M = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, en donde*

| | |
|---|---|
| $e_1 = ((g(a) \rightharpoonup h(a)), \{p_4\}, E)$ | $e_4 = ((f(a) \rightharpoonup g(a) \rightharpoonup h(a)), \{p_4\}, A)$ |
| $e_2 = ((h(b) \rightharpoonup p(b)), \{\}, M)$ | $e_5 = ((g(a) \rightharpoonup h(a) \rightharpoonup p(a)), \{\}, M)$ |
| $e_3 = ((l(b, a) \rightharpoonup p(b, a)), \{\}, M)$ | $e_6 = ((f(a) \rightharpoonup g(a) \rightharpoonup h(a) \rightharpoonup p(a)), \{\}, M)$ |

*Note que $e_2$, $e_3$, $e_5$ y $e_6$ corresponden a errores de* missing page*, mientras que $e_1$ es un error existencial y $e_4$ un error universal.*

*Los órdenes parciales $\preceq_{inf}$ y $\preceq^{sup}$, de las Definiciones 5.5.3 y 5.5.4 respectivamente, establecen los siguientes subconjuntos de errores*

$$\preceq_{inf} : \{e_1 \preceq_{inf} e_5\}; \{e_2\}; \{e_3\}; \{e_4 \preceq_{inf} e_6\}$$
$$\preceq^{sup} : \{e_4 \preceq^{sup} e_1\}; \{e_2 \preceq^{sup} e_3\}; \{e_5 \preceq^{sup} e_6 \preceq^{sup} e_3\}$$

*A continuación describiremos cómo aplicar las estrategias* reduce-delete-actions *y* reduce-insertion-actions *definidas en las secciones 5.5.2 y 5.5.2 respectivamente.*

**Estrategia *reduce-delete-actions***. *Se aplica la operación repairByDelete a cada error* minimal *w.r.t. $\preceq_{inf}$ en sitio Web $W$:*

$$W' = repairByDelete(e_4, repairByDelete(e_3, repairByDelete(e_2, repairByDelete(e_1, W))))$$

*obtenemos:*

$$W' = \{p_1, p_2, p_3\}$$
$$p_1 = m(s(b))$$
$$p_2 = m(m(\ ))$$
$$p_3 = m(\ )$$
$$-p_4 \textit{ fue eliminada}-$$

**Estrategia *reduce-insertion-actions***. *Para aplicar esta estrategia se siguen tres pasos:*

- Paso 1. *Obtener la secuencia $\mathsf{C}_{\mathbb{E}_M}(\mathbb{E}_M, \preceq^{sup})$*

$$\mathsf{C}_{\mathbb{E}_M} = [c_1, c_2, c_3] = [\{e_5, e_6, e_3\}, \{e_2, e_3\}, \{e_4, e_1\}]$$

- Paso 2. *Realizar la partición $\Gamma(\mathbb{E}_M)$*

$$\Gamma(\mathbb{E}_M) = \{m_1, m_2, m_3\}$$
$$\textit{donde}\ \ m_1 = c_1 = \{e_5, e_6, e_3\},$$
$$m_2 = c_2 \setminus c_1 = \{e_2\}\ y$$
$$m_3 = c_3 \setminus c_2 \setminus c_1 = \{e_4, e_1\}$$

- Paso 3. *Aplicar el procedimiento repairByInsert (descrito en el Algoritmo 1) a cada conjunto $m \in \Gamma(\mathbb{E}_M)$:*

$$W' = repairByInsert(m_3, repairByInsert(m_2, repairByInsert(m_1, W))))$$

*obtenemos*

$$\begin{aligned} W' &= \{p_1, p_2, p_3, p_4, p_5, p_6\} \\ p_1 &= m(s(b), f(a)) \\ p_2 &= m(m(g(a))) \\ p_3 &= m(l(b, a)) \\ p_4 &= h(b, a) \\ p_5 &= p(b, a) \\ p_6 &= p(b) \end{aligned}$$

# 6

# Un marco genérico abstracto para la verificación de sitios Web

Teniendo en cuenta la teoría de interpretación abstracta, en este capítulo se presenta una metodología abstracta de verificación que es una aproximación correcta de la metodología de verificación presentada anteriormente, en donde el dominio concreto y los operadores son reemplazados por sus correspondientes versiones abstractas.

En primer lugar, en la Sección 6.1, introducimos el concepto de interpretación abstracta. En la Sección 6.2, describimos algunos trabajos relacionados y nuestra propuesta de transformación *source-to-source*. En la Sección 6.3, describimos como representar un sitio Web. La Sección 6.4 introduce la idea clave de nuestro método, el cual es dado por un algoritmo de compresión que reduce drásticamente el tamaño de los documentos Web. Finalmente, la Sección 6.7 formaliza la noción de sistema de reescritura de términos abstracto, e ilustra como la técnica de verificación de sitios Web original [Alpuente et al., 2006a], puede aproximarse transformándola directamente al nivel abstracto.

## 6.1. Interpretación abstracta

Si consideramos una página Web como cualquier objeto que se puede referenciar con una URL (dirección) usando el protocolo HTTP, entonces la cantidad de información que se encuentra en un sitio Web es finita, pero el número (potencial) de páginas Web diferentes es infinito y a efectos prácticos suele ser imposible navegar el sitio de forma completa. Esto se debe a la secuencia infinita de páginas

Web que se pueden ir generando de manera dinámica. Por lo tanto, se necesita seleccionar un subconjunto significativo de dichas páginas, basándose en algún criterio. La teoría de la interpretación abstracta nos permite definir una aproximación para este problema.

Mostraremos el concepto de interpretación abstracta mediante un ejemplo intuitivo de la vida real. Consideremos las personas que se encuentran en una sala de conferencias. Si queremos probar que una persona no se encuentra presente en la sala, un método concreto es revisar la lista de los asistentes y, con su número de DNI (suponiendo que dos personas no tienen el mismo número de DNI), se puede probar la presencia o ausencia de una persona observando la lista de los asistentes. Sin embargo, en la lista puede estar registrado sólo el nombre de los asistentes. Si el nombre de la persona no se encuentra en la lista podemos concluir con seguridad que la persona no está, pero si el nombre está, nuestra conclusión no puede ser definitiva. Esto se debe a que existe la posibilidad de que dos personas tengan el mismo nombre.

Note que esta falta de precisión en la información puede ser adecuada para muchos propósitos. Volviendo a nuestro ejemplo, nosotros no podemos decir que una persona "sí está en la sala", todo lo que se puede decir es que "probablemente esté presente". Si la persona que estamos buscando es un criminal, esta situación puede lanzar una "alarma", pero está claro también que puede ser una "falsa alarma".

Otra situación sería si nosotros estamos interesados en información específica. Por ejemplo "si una persona de $n$ años se encuentra en la sala". En este contexto, sería innecesario mantener la lista de los nombres y las fechas de nacimientos de todos los presentes en la sala. Se puede registrar solamente la edad de la persona de más años $M$ y la de menor años $m$. De esta manera, si la edad buscada es menor que $n$ o mayor que $M$, podemos responder con seguridad que no se encuentra. En otro caso, diremos que "no lo sabemos".

En el caso de los sistemas de software, concretamente, la información precisa es en general no computable en tiempo y memoria finito. La abstracción se usa para simplificar los problemas a problemas manejables por soluciones automáticas. El punto crucial es disminuir la precisión para hacer los problemas manejables mientras se mantiene la suficiente precisión para responder a cuestiones interesantes.

La teoría de la interpretación abstracta *(abstract interpretation)* [Cousot and Cousot, 1977], provee un marco formal para desarrollar herramientas avanzadas para el análisis de flujo de datos, formalizando la idea de "computación aproximada" en la cual una computación se realiza con "descripciones de datos" en lugar de con los mismos datos . La interpretación abstracta es una teoría de aproximación semántica que se utiliza para proporcionar estáticamente respuestas válidas a cuestiones sobre el comportamiento en la ejecución de sistemas software. Los operadores semánticos son reemplazados por operadores abstractos que aproximan de forma "segura" los operadores estándar.

## 6.2. Interpretación abstracta aplicada al análisis de sitios Web

En la literatura, la interpretación abstracta ha sido escasamente aplicada al análisis de sitios Web. Actualmente, muy pocos trabajos tratan este problema, y todos ellos se focalizan sobre aspectos dinámicos de los sistemas distribuidos subyacente en lo sitios Web. Por ejemplo, en [Legall et al., 2006] se desarrolla una aproximación abstracta que permite analizar los protocolos de comunicación de un sistema distribuido particular con el objetivo de cumplir un comportamiento global correcto del sistema. En [Kadhi and El-Gendy, 2006], se usa interpretación abstracta para verificar propiedades de seguridad: la metodología se aplica a un conjunto de descripciones abstractas de *input/output* en una máquina de estados finitos con el objetivo de validar protocolos criptográficos que implementan transacciones seguras en la Web.

De acuerdo con nuestro conocimiento, en este trabajo se presenta la primera metodología abstracta que soporta la verificación de aspectos estáticos como así también dinámicos de sitios Web.

Nuestra inspiración viene del área de aproximación de consultas XML (*approximate XML query answering*) [Buneman et al., 2003; Polyzotis et al., 2004], donde las consultas XML son ejecutadas sobre versiones comprimidas de datos XML (ej. sinopsis de documentos) con el objetivo de obtener rápidamente respuestas aproximadas. De forma general, la sinopsis de documentos representa la abstracción del dato original sobre el cual se ejecuta la computación abstracta

(ej., consultas).

En este capítulo, describimos una aproximación usando interpretación abstracta para verificar sitios Web, la cual mejora en gran medida las prestaciones de la herramienta de verificación previa. Proveemos un esquema de abstracción donde el sitio Web, como así también las reglas de la especificación, son transformados en construcciones del lenguaje original. También comprobamos las condiciones que aseguran la corrección de la aproximación, de modo que el motor de reescritura abstracto resultante soporte con seguridad la verificación Web. Definimos nuestro marco de forma paramétrica respecto de la abstracción considerada y, por otro lado, caracterizamos las condiciones que hacen posible implementar la abstracción por medio de una transformación *source-to-source* del sitio y la especificación Web a su versión abstracta. Gracias a esta aproximación *source-to-source*, podemos adaptar y reusar fácilmente todas las facilidades soportadas por nuestro sistema de verificación previo. Como un mejora adicional, nuestra metodología de verificación abstracta permite potencialmente considerar el tratamiento de sitios Web con un número infinito de páginas.

## 6.3. Representación de sitios Web

Para describir un sitio Web, veremos la representación dada en [Lucas, 2005]. Usamos un alfabeto $\mathcal{P}$ para dar nombre a las páginas Web, así como para expresar la diferentes trasiciones entre ellas.

**Definición 6.3.1 (sucesor inmediato)** *La relación de* sucesor inmediato *(*immediate successors*) para una página Web $p$ se define como*

$$\rightarrow_p = \{(p, p') \subseteq \mathcal{P} \times \mathcal{P} \mid p' \textit{ es directamente accesible a partir de } p\}$$

La Definición 6.3.1 establece la relación abstracta entre las páginas $p$ y sus sucesores inmediatos (ej., las páginas $p_1, \ldots, p_n$ las cuales son apuntadas desde $p$ por medio de *hyperlinks*).

El par $(\mathcal{P}, \rightarrow_{\mathcal{P}})$, donde $\rightarrow_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} \rightarrow_p$, es un *Sistema de reducción abstracta* (*Abstract Reduction System*) (ver ARS [Baader and Nipkow, 1998], Capítulo 2). Usaremos la relación computacional asociada $\rightarrow_{\mathcal{P}}$, $\rightarrow^+_{\mathcal{P}}$, etc., para describir el

comportamiento dinámico de un sitio Web. En este contexto, la alcanzablidad de una página Web $p'$ desde otra página $p$ puede ser expresada como $p \rightarrow^*_{\mathcal{P}} p'$.

**Definición 6.3.2 (Sitio Web)** *Un* sitio Web *(*Web site*) queda definido como el conjunto de páginas alcanzables desde una página inicial, y se denota por*

$$W = \{p_1, \ldots, p_n\}, s.t.\ \exists i, 1 \leq i \leq n, s.t.\ \forall j, 1 \leq j \leq n,\ p_i \rightarrow^*_W p_j$$

La Definición 6.3.2 formaliza la idea de que un sitio Web tiene una página inicial desde la cual se puede visitar el sitio Web entero. Diremos que $p$ es una *página Web principal* (*main Web page*) de un sitio Web $W$, si $p \in W$ y $\forall p' \in W,\ p \rightarrow^*_W p'$. Note que en un sitio Web pueden existir mas de una página Web principal.

## 6.4. Compresión de la Web

Cuando navegamos por un sitio Web, es común encontrar un número de páginas que tienen una estructura similar pero con diferentes contenidos. Esto pasa muy a menudo cuando las páginas son generadas dinámicamente con información que proviene de una base de datos (ej., el sitio Web de Amazon). Esta situación, puede hacer impracticable nuestro análisis, al menos que seamos capaces de proveer un mecanismo para reducir drásticamente la talla del sitio Web. En este sentido, a continuación definimos una transformación que nos permite reducir el número de ramas de un término.

Primero, describimos dos funciones auxiliares necesarias para nuestra transformación. Sean $s, t \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$. Denotamos por $root(s)$ al símbolo de función en la raíz de $s$, en símbolos tenemos que

$$root(f(s_1, \ldots, s_n)) = f$$

La función $join(s, t)$ devuelve el término que se obtiene al combinar sus argumentos de $s$ y $t$ tal que $root(s) = root(t)$ (descartando los duplicados). Formalmente

$$join(f(s_1,..,s_n), f(t_1,..,t_m)) = f(dd([s_1,..,s_n,t_1,..,t_m]))$$

donde $dd(l) = a_1,\ldots,a_n$ siempre que $drop\_duplicates(l) = [a_1,\ldots a_n]$ y

$$drop\_duplicates([x]) = [x]$$

$$drop\_duplicates(x:xs) = \begin{array}{ll} \text{if } member(x,xs) & \text{then } drop\_duplicates(xs) \\ & \text{else } x : drop\_duplicates(xs) \end{array}$$

**Definición 6.4.1 (compresión de términos)** *Sea* $t = f(t_1,...,t_n) \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$*. Entonces, el término* $t$ *se comprime juntando los sub-términos que tienen el mismo símbolo raíz. La compresión de términos se describe por medio del Algoritmo 2.*

**Algorithm 2** Función de compresión de un término.

**Input:**
Term $t = f(t_1,\ldots,t_n)$

**Output:**
Term $f(t'_1,\ldots,t'_m)$, with $m \leq n$

```
1: function COMPRESS (t)
2:   if n = 0 then
3:     ← f
4:   else if ∃ i, j s.t. root(t_i) = root(t_j), then
5:     t' ← join(t_i, t_j)
6:     ← COMPRESS(f(t_1, ..., t_{i-1}, t', t_{i+1}, ..., t_{j-1}, t_{j+1}, ..., t_n))
7:   else
8:     ← f(COMPRESS(t_1), ..., COMPRESS(t_n))
9:   end if
10: end function
```

La idea que subyace a la Definición 6.4.1 se ilustra en la Figura 6.1. Intuitivamente, primero se unen todos los argumentos con el mismo símbolo raíz ocurriendo al mismo nivel $i$, luego, la compresión procede de manera recursiva al nivel $(i+1)$. La Figura 6.1(b) muestra la compresión del término de la Figura 6.1(a).

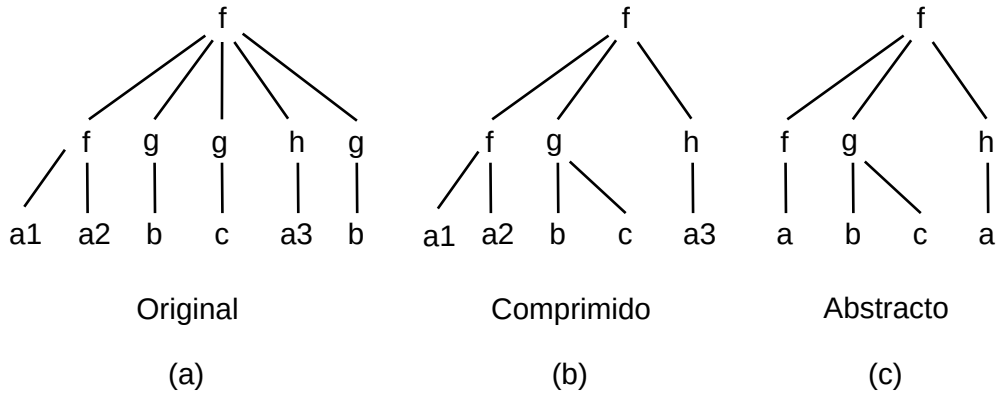


Figura 6.1: Compresión y abstracción un término

## 6.5. Especificación Web abstracta

Primero introduciremos la noción de dominio abstracto y de sistema de reescritura de términos abstracto. Nuestra primera definición está justificada por el hecho de que buscamos formalizar la abstracción como una transformación *source-to-source* que permita transformar los documentos Web y las reglas en construcciones del lenguaje original y, de esta manera, hacer que coincidan los dominios concreto y abstracto, e implementar los operadores abstractos haciendo uso de los constructores también.

**Definición 6.5.1 (álgebra de términos *non-ground* abstracta, poset)** *Sea* $\mathcal{D} = (\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V}), \leq)$ *el dominio estándar de (clases de equivalencia de) términos ordenados por el orden parcial estandar* $\leq$ *inducido por el pre-orden de términos dados por la relación de ser "más general". Entonces, el dominio de términos abstractos* $\mathcal{D}^\alpha$ *es igual a* $D$*.*

Definimos la abstracción ($t^\alpha$) de un término $t$ como: $t^\alpha = \alpha(t)$. Nuestro marco es paramétrico respecto a la función de abstracción $\alpha$, la cual puede ser usada para ajustar la precisión de la aproximación.

**Definición 6.5.2 (función de abstracción de términos $\alpha$)**

$$\alpha :: \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V}) \rightarrow \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$$
$$\alpha(t) = \alpha\_(\epsilon, t)$$

*donde la función auxiliar $\alpha_{\_}$ está dada por*

$$\begin{aligned} \alpha_{\_}(\_, x) &= x, \text{ if } x \in \mathcal{V} \\ \alpha_{\_}(c, f(t_1, \ldots, t_n)) &= f(\alpha_{\_}(c.f, t_1), \ldots, \alpha_{\_}(c.f, t_n)), \text{ if } f \in \mathcal{T}ag \\ \alpha_{\_}(c, w) &= atext_{\_}(c, w), \text{ if } w \in \mathcal{T}ext \end{aligned}$$

Note que en la Definición 6.5.2, los elementos de $\mathcal{T}ext$ se abstraen tomando en cuenta la cadena de *tags* bajo la cual aparece una pieza de texto. Esto se formaliza por medio de la función genérica

$$atext_{\_} : \mathcal{T}ag^* \times \mathcal{T}ext \to \mathcal{T}ext$$

Actualmente, la función *atext* se deja sin definir y es un parámetro formal de la función de abstracción de términos $\alpha$, y puede ser dada para ajustar la abstracción de cada dominio particular. Por ejemplo, en el caso donde no sea necesaria una distinción entre textos, cada elemento en *text* podría ser reemplazado por algún símbolo de constante fresco $C$.

**Ejemplo 6.5.3** *Consideremos nuevamente el término $t$ de la Figura 6.1(b), y $\alpha(w) = first(show\ w)$, donde $first(x : xs) = x$, la abstracción $\alpha(t)$ se muestra en la Figura 6.1(c).*

En dominios específicos, por ejemplo, una función de abstracción para librerías digitales podría requerir discriminar el texto que aparece debajo de los *tags* que corresponden con libros o revistas, mientras que la demás información, que se encuentra a niveles más profundos dentro de la página, podría no ser relevante. En otro ejemplo, para un sistema de ventas *online*, se podría definir una función de abstracción que distinga la información entre clientes y artículos.

Con el objetivo de definir una especificación Web abstracta (*abstract Web specifications*), necesitamos definir reglas de reescritura abstractas (*abstract rewrite rules*) — regla de corrección y completitud, ver Sección 3.2 —.

**Definición 6.5.4 (abstract rewrite rule)** *Sea $w_M \equiv l \rightharpoonup r$ una regla de completitud (resp. $w_N \equiv l \rightharpoonup error$ una regla de corrección). Denotamos por $w_M^\alpha$ (resp. $w_N^\alpha$) la abstraccón de $w_M$ (resp. $w_N$), donde $w_M^\alpha = \alpha(l) \rightharpoonup \alpha(r)$ (resp. $w_N^\alpha = \alpha(l) \rightharpoonup error$).*

Cuando no exista confusión, escribiremos $w_M^\alpha = l^\alpha \rightharpoonup r^\alpha$ (resp. $w_N^\alpha = l^\alpha \rightharpoonup error$). De esta manera, una especificación Web $(I_N, I_M, R)$ se abstrae — de forma natural — a $(I_N^\alpha, I_M^\alpha, R)$.

## 6.6. Abstracción del sitio Web

Después de aplicar la transformación abstracta dada en la Sección previa, el número de páginas de un sitio Web puede ser reducido significativamente. Por ejemplo, consideremos $p_1, p_2 \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ s.t. $p_1 = professor(name(P.Almodovar))$ y $p_2 = professor(name(P.Cruz))$, consideremos también la función de abstracción $\alpha$ dada en el Ejemplo 6.5.3, entonces $\alpha(p_1) = \alpha(p_2) = professor(name(P))$.

Al mismo tiempo, la función de compresión de términos dada en la Sección 6.4 contribuye aún más a reducir el tamaño de cada página Web (ver Figura 6.1).

Por otro lado, para visitar o navegar un sitio Web (considerando la Definición 6.3.2), comenzamos desde una página inicial y aplicamos de manera recursiva la relación de sucesor ($\rightarrow$). Utilizando un algoritmo *Depth-first search* (DFS) [Cormen et al., 2001], se puede aproximar un sitio Web como sigue.

$$
\begin{array}{l}
dfs\ (p,\ W^\alpha) = \\
\qquad p^\alpha \leftarrow compress(\alpha(p)) \\
\qquad W^\alpha \leftarrow W^\alpha \cup \{p^\alpha\} \\
\qquad \forall i \text{ s.t. } (p, p_i) \in \rightarrow_p \text{ and } compress(\alpha(p_i)) \notin W^\alpha \\
\qquad\qquad W^\alpha \leftarrow dfs(p_i,\ W^\alpha) \\
\qquad \leftarrow W^\alpha
\end{array}
$$

Ahora estamos listos para definir nuestra noción de abstracción de sitios Web.

**Definición 6.6.1 (Sitio Web abstracto)** *Sea $W$ un sitio Web y $p$ la página inicial de $W$. Entonces, la abstración de $W$ se define como:*

$$\alpha(W) = dfs(p, \emptyset)$$

En un sitio Web, el número de *links* de una página es finito, pero la profundidad de las ramas del grafo puede ser infinito. La condición dada en el algoritmo *dfs* nos sirve para podar las ramas del grafo y, junto con la Definición 6.6.1, obtener una representación finita abstracta de los sitios Web.

## 6.7. Verificación de sitios Web abstractos

La función de absracción dada en la Definición 6.5.2 define la abstracción mediante una transformación *source-to-source*. Gracias a este esquema de aproximación *source-to-source*, todas las facilidades soportadas por nuestro sistema de verificación previo pueden ser adaptadas y reusadas, de manera directa, con poco esfuerzo.

Informalmente, nuestra metodología de verificación abstracta se aplica al sitio Web abstracto y a la especifiación abstracta. Dado un sitio Web $W$ y una especificación $(I_N, I_M, R)$, primero generamos las correspondientes abstracciones $W^\alpha$ y $(I_N^\alpha, I_M^\alpha, R)$. Luego — explotando la transformación *source-to-source* — aplicamos nuestro algoritmo de verificación [Alpuente et al., 2006a] para analizar $W^\alpha$ respecto a $(I_N^\alpha, I_M^\alpha, R)$.

**Definición 6.7.1** *Sea $W$ un sitio Web y $(I_N, I_M, R)$ una especificación Web, y sean $W^\alpha$ y $(I_N^\alpha, I_M^\alpha, R)$ sus correspondientes versiones abstractas Llamamos error de corrección (resp. completitud) abstracto, a cada error de corrección (resp. completitud) detectado en $W^\alpha$ utilizando $(I_N^\alpha, I_M^\alpha, R)$.*

Para garantizar la validez del proceso abstracto, es necesario asegurar que, al trabajar con datos abstractos, la relación de reescritura parcial $\rightharpoonup$ aproxima correctamente el comportamiento de la relación de reescritura parcial sobre la correspondiente representación concreta.

En otras palabras, si un término (concreto) $t_1$ se reescribe parcialmente a un requerimiento $t_2$ utilizando la regla $r$, y $\alpha(t_1)$ se reescribe parcialmente a $t'$ utilizando $r^\alpha$, entonces $\alpha(t_2) \preceq t'$. En símbolos tenemos

$$t_1 \rightharpoonup_r t_2 \wedge \alpha(t_1) \rightharpoonup_{r^\alpha} t' \Rightarrow \alpha(t_2) \preceq t' \tag{6.1}$$

Cuando esto pasa, decimos que la abstracción $\alpha$ (o mas precisamente la función $\alpha\_(c, f)$ que es un parámetro de la transformación de abstracción) es segura (*safe*) respecto a $(I_N, I_M, R)$ y $W$. Del mismo modo, para asegurar el punto fijo en el dominio abstracto, exigimos que un paso de la reescritura parcial en el dominio abstracto corresponda con un paso de reescritura parcial en el dominio concreto. Bajo este supuesto, la siguiente proposición puede ser probada fácilmente y, de esta manera, establecer la validez de la metodología de verificación abstracta.

**Proposición 6.7.2** *Sea* $(I_N, I_M, R)$ *una especificación Web y* $W$ *un sitio Web. Sean respectivamente* $(I_N^\alpha, I_M^\alpha, R)$ *y* $W^\alpha$ *las correspondientes versiones abstractas de* $(I_N, I_M, R)$ *y* $W$*. Si la abstracción* $\alpha$ *es segura respecto a* $(I_N, I_M, R)$ *y* $W$ *entonces cada error de corrección (resp., completitud) abstracto señala un error de corrección (resp., completitud) respecto a* $(I_N, I_M, R)$ *y* $W$*.*

Para concluir, esta aproximación formalizada nos permite aplicar el marco de verificación original y, al mismo tiempo, obtener un equilibrio conveniente entre la eficiencia y la precisión del análisis para cada dominio específico.

# 7

# Implementación y evaluación experimental

En este capítulo, describimos la implementación de nuetro prototipo WebVerdi-M, la arquitectura del prototipo WebVerdi-M y los resultados experimentales obtenidos al utilizar WebVerdi-M con grandes volúmenes de datos.

## 7.1. Prototipo WebVerdi-M

Junto con el avance de la línea de investigación, se desarrolló WebVerdi-M. WebVerdi-M es una herramienta que, mediante técnicas de reescritura, permite especificar condiciones de integridad de un sitio web y diagnosticar los errores de un portal web computando los requerimientos no satisfechos por el sitio.

Con el objetivo de tener una buena interacción con el usuario y la posibilidad de interactuar con otras herramientas, hemos implementado una interfaz Web amigable para el usuario y un servicio web que permite que otras herramientas o usuarios puedan usar sus operaciones.

La aplicación ha sido estructurada como un servicio Web SOAP *(Service Oriented Architecture Paradigm)* [World Wide Web Consortium, 2003]. En este paradigma, los servicios son distribuidos, autónomos e independientes. Ellos se realizan utilizando protocolos estándar, con el objetivo de construir redes de aplicaciones colaborativas.

Al tratarse de una arquitectura orientada a servicios,WebVerdi-M permite acceder al nucleo del motor de verificación Verdi-M como una entidad reusable. La implementación está públicamente disponible en la siguiente url:

`http://www.dsic.upv.es/users/elp/webverdi-m` .

WebVerdi-M se puede dividir en dos capas: *front-end* y *back-end*. La capa *back-end* proporciona el servicio Web que da soporte a la capa *front-end*. Esta arquitectura permite que usuarios en la red puedan invocar las funcionalidades del servicio Web a través de las interfaces disponibles.

La herramienta consiste de los siguientes componentes: el servicio Web WebVerdiService, el cliente Web WebVerdiClient, el motor Verdi-M, una XML API, y la base de datos DB. En la Figura 7.1 ilustramos la arquitectura del sistema.

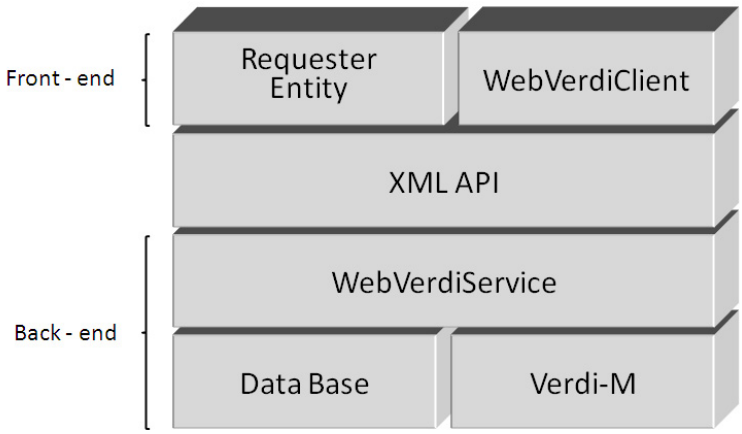


Figura 7.1: Componentes de WebVerdi-M

### WebVerdiService

El servidor se estructura como un servicio Web implementado en Java 1.4 [Sun Microsystems, ]. El servicio Web manipula objetos persistentes para permitir reducir el tráfico en la red. La persistencia se implementa utilizando el paquete TriActive JDO [SourceForge, 2005]. TriActive JDO es una implementación de código libre *(open source)* de la especificación JDO de Sun [Sun Microsystems, 2006][1].

Todo el proceso de verificación se ejecuta del lado del servidor por los módulos escritos en Maude. Las ventajas de utilizar Maude están descritas en el Sección

[1]JDO se diseñó para soportar de manera transparente cualquier base de datos que cumpla con la especificación JDBC.

4.2, y son básicamente las propiedades de asociatividad y conmutatividad que permiten una implementación del algoritmo de simulación simple y elegante.

El servicio Web exporta seis operaciones que son accesibles a través de mensajes estándares XML. Estas operaciones permiten: almacenar un sitio Web, eliminar un sitio Web, recuperar un sitio Web, añadir una página a un sitio Web, verificar la corrección y verificar la completitud. El servicio Web actúa como un "simple punto de acceso" al motor Verdi-M, el cual implementa la metodología de verificación Web en Maude. Siguiendo los estándares, la arquitectura es independiente de la plataforma y del lenguaje; también la aplicación se puede acceder vía *script* así como desde alguna otra aplicación cliente por medio de la red.

### XML API

Para una correcta comunicación entre el WebVerdiService y el WebVerdiClient (o cualquier usuario), se debe acordar un formato común para los mensajes que serán entregados, esto permite que puedan ser correctamente interpretados por cada una de las parte. El servicio Web WebVerdiService se desarrolló respetando una API (ver [Alpuente et al., 2007e]) que está en sitonía con la librería del motor Verdi-M. En el desarrollo se utilizó Oracle JDeveloper, incluyendo el generador de WSDL *(Web Services Description Language)* para hacer la API disponible. El servidor OC4J (el servidor Web integrado en Oracle JDeveloper) administra los procesos comunes al web Service. Los errores detectados también son codificados cómo documentos XML. El detalle de la especificación de la API así como la descripción de los documentos XML se puede consultar en [Alpuente et al., 2007e].

### Verdi-M

Verdi-M es la parte mas importante de la herramienta. Aquí es donde se implementa la metodología de verificación. Este módulo está implementado en el lenguaje Maude y es independiente del resto de los módulos del sistema. Las operaciones de Verdi-M se invocan desde el módulo WebVerdiService. Una descripción a bajo nivel de Verdi-M se puede encontrar en [Alpuente et al., 2007a].

### WebVerdiClient

El principal objetivo, en el desarrollo del cliente Web, fue proveer una interfaz *intuitiva* y *amigable* para el usuario. WebVerdiClient consiste de una interfaz gráfica Java que permite utilizar las funcionalidades ofrecidas por el servicio Web. El cliente utiliza la API especificada (ver [Alpuente et al., 2007e]) para interactuar con el servicio Web y la ejecución se realiza por medio de Java Web Start.

WebVerdiClient está provisto de una interfaz que permite manipular la especificación Web y las páginas de un sitio Web. Sobre la especificación Web se puede trabajar con tres representaciones diferentes: (i) en el primer caso, la idea de acceder a la regla utilizando una representación de árboles de tipo "directorio", lo cual permite una adaptación rápida por parte del usuario, esto se debe a la familiaridad que se tiene con este tipo de estructuras; (ii) la segunda representación se basa en una estructura XML; (iii) en el tercer caso, se puede se puede utilizar la sintaxis de la propia álgebra de términos. En la herramienta se puede trabajar de manera indistinta en cualquiera de las tres vistas. En la Figura 7.2, se muestra una captura de pantalla del cliente Web WebVerdiClient. Respetando la API definida, un usuario podría utilizar cualquier otro cliente para trabajar.

### DB

En la interacción entre el servicio Web y la aplicación cliente se necesita transferir abundante información. Consideremos el siguiente escenario donde un usuario modifica una regla de la especificación Web y luego verifica los errores en un sitio Web. Esto trae aparejado que, en cada modificación, sea necesario enviar la especificación modificada junto con todo el sitio Web a ser verificado desde la aplicación cliente hacia el servicio Web. En el siguiente paso, el proceso de verificación detecta y transfiere los errores desde el servicio Web hacia la aplicación cliente.

La arquitectura de servicios Web estándar requiere que las aplicaciones clientes esperen hasta que los datos se reciban y se devuelvan los resultados. En nuestro contexto, esta arquitectura causaría una significante sobrecarga en la red y en el tiempo en la aplicación. Para evitar esto y proveer una mejor prestación al usuario, utilizamos del lado del servidor una base de datos local (*MySql)* que almacena temporalmente el sitio Web así como los errores detectados.

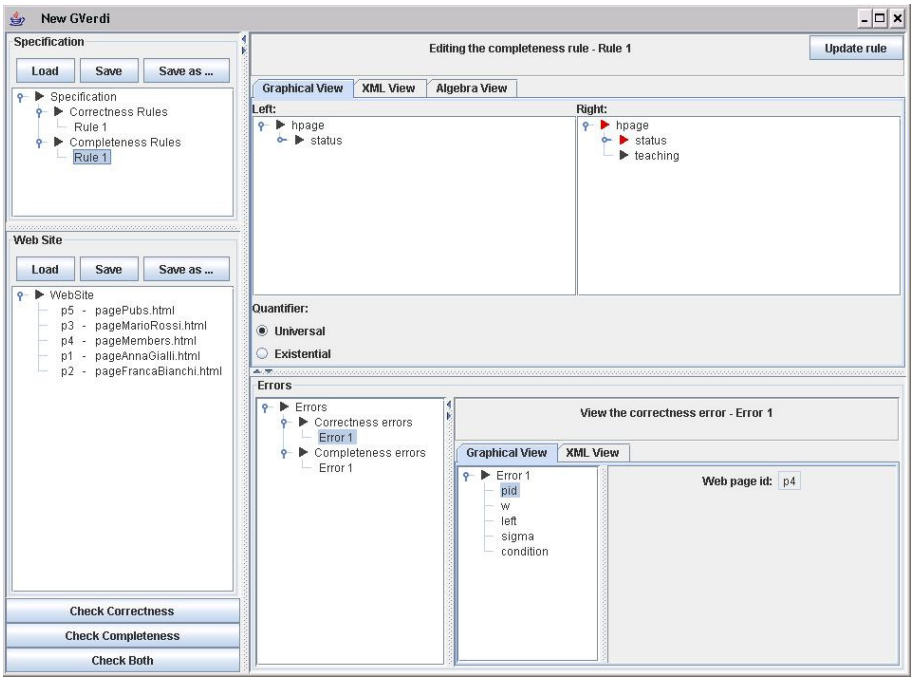


Figura 7.2: Captura de pantalla del cliente Web WebVerdiClient

## 7.2. Evaluación experimental

Con el objetivo de evaluar las prestaciones de Verdi-M en sistemas de tamaño real, es decir, con grandes volúmenes de datos que excedan los ejemplos presentados, realizamos diferentes evaluaciones experimentales utilizando distintas reglas de corrección y de completitud sobre un número significativo de documentos XML. Los documentos XML fueron generados utilizando el "generador de documentos XML" xmlgen, disponible dentro del proyecto XMark [Centrum voor Wiskunde en Informatica, 2001]. Para la generación de los documentos se utilizaron diferentes factores de escala.

En la Tabla 7.1, se muestran los resultados obtenidos para la simulación de tres especificaciones Web diferentes $WS1$, $WS2$ y $WS3$ en cinco documentos XML generados. Especificamente, los factores de escala utilizados van desde 0,01 a 0,1 y generan documentos XML cuyo rango va desde 1Mb — correspondiente a un documentos XML de alrededor 31.000 nodos — a 10Mg — correspondiente a un documentos XML de 302.000 nodos.

Las especificaciones Web $WS1$ y $WS2$ comprueban el poder de verificación

de nuestra herramienta para reglas de corrección. La especificación Web $WS2$ es más compleja que la especificación Web $WS1$, esto se debe a que las reglas son más restrictivas y contienen evaluación de funciones. La especificación $WS3$ verifica la completitud de los documentos XML. Los resultados que se muestran en la Tabla 7.1 fueron obtenidos en una computadora personal equipada con un Pentium Centrino CPU de 1.75 GHz, 1Gb de memoria RAM y 40Gb de disco duro ejecutando el sistema operativo Ubunto Linux 5.10.

| **Size** | **Nodes** | **Scale factor** | **Time** | | |
|---|---|---|---|---|---|
| | | | $WS1$ | $WS2$ | $WS3$ |
| 1 Mb | 30,985 | 0,01 | 0,93 s | 0,96 s | 165,57 s |
| 3 Mb | 90,528 | 0,03 | 2,60 s | 2,84 s | 1,768,74 s |
| 5 Mb | 150,528 | 0,05 | 5,97 s | 5,94 s | 4,712,15 s |
| 8 Mb | 241,824 | 0,08 | 8,60 s | 9,42 s | 12,503,45 s |
| 10 Mb | 301,656 | 0,10 | 12,45 s | 12,64 s | 21,208,49 s |

Cuadro 7.1: Tiempos de ejecución utilizando Verdi-M

A continuación, comentaremos los resultados obtenidos. Con respecto a los tiempos en la verificación de la corrección, estos son extremadamente eficientes y el crecimiento es lineal con respecto al número de nodos de los documentos. En la Tabla 7.1, observamos que los tiempos de ejecución son cortos para documentos muy largos (e.g. ejecutando las reglas de corrección de la especificación Web $WS1$ sobre un documento de 10Mg con 302,000 nodos se tardó menos de 13 segundos). Considerando la verificación de la completitud vemos que el tiempo invertido en la computación del punto fijo, que se requiere para la evaluación de las reglas de completitud (ver [Alpuente et al., 2006a]), excede el tiempo deseado y, en este caso, somos capaces de procesar eficientemente documentos XML cuyo tamaño no sea mayor que 1Mg (la ejecución de las reglas de completitud de la especificación $WS3$ sobre documentos XML de 1Mg con 31,000 nodos toma menos de 3 minutos).

Finalmente, queremos destacar que el comportamiento de nuestra implementación en Maude del sistema de verificación excede en gran medida a nuestro sistema previo, llamando GVErdi [Alpuente et al., 2006a; Ballis and Vivó, 2005], el cual sólamente fue capaz de procesar pequeños repositorios XML (no mayores a 1Mg) dentro de un tiempo razonable.

En la Capítulo 6 de esta tesis se presenta un trabajo el cual, utilizando la técnica de *interpretación abstracta*, permite acercar el tiempo de la verificación de la completitud al tiempo de la verificación de la corrección. Actualmente, estamos trabajando para realizar una evaluación más exhaustiva de esta nueva técnica.

# 8

# Conclusiones

Los sistemas software para la web se han convertido en un instrumento insustituible de la moderna sociedad de la información: hacen posible el intercambio de información de forma rápida y a escala global; constituyen el medio natural de las grandes transacciones financieras; permiten acceder de forma rápida y selectiva a grandes volúmenes de información especializada, etc. Esto ha incrementado la complejidad de los sitios web y puesto de manifiesto la necesidad de asistir a los administradores de sistemas web en la reparación de las posibles incorrecciones o consistencias. Es esencial el desarrollo de métodos que se apliquen a la verificación de sitios y servicios web que permitan no sólo detectar posibles errores en los enlaces o en la estructura sino también en la semántica de los mismos, y de esta manera poder aplicar estrategias de reparación para obtener sitios web que sean correctos y completos.

El origen de este trabajo está dado en [Alpuente et al., 2006a; Ballis, 2005], donde se presenta un marco para la verificación de sitios Web. Dicho marco permite especificar restricciones de integridad para un sitio Web y detectar con precisión las partes que no satisfacen la especificación. Una vez realizada la verificación resulta interesante poder corregir las anomalías detectadas.

Los resultados obtenidos en esta línea se obtuvieron durante el primer periodo de esta tesis, donde ampliamos esta línea de investigación definiendo un nuevo motor algebráico utilizando el lenguaje *Maude* [Alpuente et al., 2007a]. Luego, mediante técnicas de estrategias de reparación adaptadas al campo de las bases de datos, se definieron estrategias que permitieron optimizar el proceso de reparación de errores con el tratamiento y eliminación de errores de completitud. Esto nos permitió una optimización efectiva del proceso de reparación de un sitio Web,

debido a que el usuario debe de reparar una cantidad menor de errores para obtener un sitio Web correcto y completo. [Alpuente et al., 2007c].

En [Alpuente et al., 2008], presentamos, de acuerdo a nuestro conocimiento, la primera metodología abstracta que considera aspectos estáticos así como dinámicos de sitios Web. La abstracción se formaliza como una transformación *source-to-source* (fuente a fuente) que es paramétrica con respecto al dominio abstracto considerado, lo que permite una traducción de los documentos Web y las reglas de la especificación en construcciones del lenguaje original. Con esta abstracción, se puede compensar el alto coste de ejecución de analizar sitios Web complejos.

Asimismo, en [Alpuente et al., 2007b; Alpuente et al., 2007d] realizamos un estudio de escalabilidad (en función del tiempo de ejecución y tamaño del sitio Web) de la metodología de verificación. Los resultados obtenidos resultan muy prometedores y muestran una gran *performance*. En la evaluación de reglas de corrección se empleó un tiempo menor a un segundo para evaluar un árbol de aproximadamente 30.000 nodos. Considerando la verificación de la completitud vemos que el tiempo invertido en la computación del punto fijo, que se requiere para la evaluación de las reglas de completitud, excede el tiempo deseado (ver [Alpuente et al., 2006a]) y, en este caso, somos capaces de procesar eficientemente documentos XML cuyo tamaño no sea mayor que 1Mg. Por ejemplo, la verificación de un especificación de completitud sobre documentos XML de 1Mg con 31,000 nodos tomó menos de 3 minutos.

Una implementación de nuestra metodología, WebVerdi-M, está disponible en

`http://www.dsic.upv.es/users/elp/webverdi-m/`.

# Parte II

# Generalización Ecuacional

# Resumen de Contribuciones de la Parte II

Un componente importante para asegurar la terminación de muchas técnicas de manipulación de programas, es el cálculo de las generalizaciones menos generales (*lggs*). En esta parte de la tesis, extendemos el algoritmo de Huet de generalización sin tipos a un algoritmo con tipos *order-sorted* que puede manejar *sorts, subsorts, y polimorfismo*. Además presentamos un algoritmo modular de generalización ecuacional, donde los símbolos de función pueden obedecer cualquier combinación de los axiomas de asociatividad, conmutatividad e identidad. Los algoritmos calculan un conjunto completo de lggs ecuacionales (módulo la teoría considerada). Además, nuestros algoritmos están expresados por medio de un sistema de inferencia para el cual damos una prueba de corrección. Este trabajo abre nuevas posibilidades para aplicaciones como evaluación parcial, síntesis de programas, demostradores de teoremas y para los nuevos lenguajes basados en reglas como ASF+SDF, Elan, OBJ, Cafe-OBJ, y Maude.

Brevemente, las principales contribuciones de los capítulos que siguen son:

- Redefinimos, mediante un sistema de inferencia, los algoritmos de cálculo de lgg existentes.
- Definimos los primeros algoritmos de cálculo de lgg para los casos *order-sorted* y *ecuacional*.
- Implementamos una nueva herramienta web, la cuál implementa el algoritmo para el caso ecuacional y está disponible públicamente.

# 9

# Introducción

El problema de garantizar la terminación mediante técnicas para la manipulación de programas surge en muchas áreas de ciencias de la computación, incluyendo análisis automático, síntesis, verificación, especialización, y la transformación de programas (ver [Gallagher, 1993; Muggleton, 1999; Boyer and Moore, 1980a; Pfenning, 1991]). Un componente importante para garantizar la terminación de estas técnicas es un algoritmo de generalización (también llamado anti–unificación) que, para un par de expresiones de entrada, devuelve su generalización menos general (LGG - *Least General Generalization*), es decir, una generalización que es más específica que cualquier otra generalización.

En unificación se generan los unificadores más generales (mgu, *most general unifier), que cuando se aplican a dos expresiones, las hace equivalentes a la instancia más general que tienen en común las expresiones de entrada [Lassez et al., 1988]. Al contrario que en unificación, donde es de interés el unificador más general (mgu), en generalización es de interés la generalización menos general (lgg,* least general generalization). Similarmente al caso de la unificación ecuacional, nos centramos en generar un conjunto completo y minimal de lggs. Este conjunto de lggs es el dual análogo de un conjunto completo y mínimo de unificadores para problemas de unificación.

Por ejemplo, en la evalución parcial (PE) de programas lógicos [Gallagher, 1993], partiendo de un conjunto inicial de llamadas, la principal idea es construir un conjunto finito (posiblemente parcial) de árboles de deducción, y entonces extraer desde estos árboles un nuevo programa $P$ que nos permite ejecutar cualquier instancia de las llamadas iniciales. Para garantizar que el programa evaluado parcialmente *cubre* todas las llamadas. La mayoría de los procedimentos de PE se

generan de forma recursiva, especializando algunas llamadas que son generadas dinámicamente durante este proceso. Esto requiere algún tipo de generalización, a fin de hacer cumplir la terminación del proceso: si una llamada en $P$ que no está lo suficientemente cubierta por el programa *embebe* una llamada que ya ha sido evaluada, ambas llamadas son generalizadas calculando el lgg de ambas. En la literatura sobre evaluación parcial, la generalización menos general se conoce también como *generalización más específica* (msg) o *la anti–instancia menos común* (lcai) [Mogensen, 2000].

El cálculo de lggs es una parte central para muchos programas de síntesis y algoritmos de aprendizaje como por ejemplo los desarrollados en el área de programación lógica inductiva [Muggleton, 1999], y también en la conjetura de lemas en la demostración de teoremas inductivos como Nqthm [Boyer and Moore, 1980b] y sus extensiones ACL2 [Kaufmann et al., 2000a].

El lgg fue introducido originalmente por Plotkin en [Plotkin, 1970], ver también [Reynolds, 1970]. De hecho, el trabajo de Plotkin se originó a partir de las consideraciones en [Popplestone, 1969] quién, puesto que la unificación es útil en la deducción automática por el método de resolución, apuntó por primera vez que su dual podría resultar útil para la inducción.

Por otro lado, muy a menudo, las aplicaciones de generalización pueden tener que llevarse a cabo en contextos en los que la función debe satisfacer ciertos *axiomas ecuacionales*. Por ejemplo, en lenguajes basados en reglas como son ASF+SDF [Bergstra et al., 1989], Elan [Borovanský et al., 2002], OBJ [Goguen et al., 2000], CafeOBJ [Diaconescu and Futatsugi, 1998], and Maude [Clavel et al., 2007] algunos símbolos de función pueden ser declarados con reglas algebraicas (los llamados *atributos ecuacionales* de OBJ, CafeOBJ y Maude), de forma que las reglas del programa se ejecutan módulo las clases de equivalencias, evitando así el riesgo de no terminación. Del mismo modo, en demostradores de teoremas, tanto generales como de primer orden y demostradores de teoremas inductivos habitualmente se generan teorías ecuacionales para algunos símbolos de función como la conmutatividad y asociatividad. En otro ámbito, en [Escobar et al., 2007; Escobar et al., 2006b] se propone el uso de reglas para aplicaciones orientadas a la verificación de protocolos de seguridad. En ellas, el análisis simbólico de accesibilidad módulo las propiedades algebraicas se usa para razonar sobre

la seguridad en los intentos de ataques a bajo nivel que explotan las propiedades algebraicas de las funciones utilizadas en el protocolo (por ejemplo, cifrado conmutativo). Un estudio de las propiedades algebraicas utilizadas en protocolos criptográficos se encuentra en [Cortier et al., 2006]. Sorprendentemente, a diferencia del caso dual de la unificación ecuacional, que si ha sido investigada a fondo (véase, por ejemplo, los trabajos [Siekmann, 1989; Baader and Snyder, 1999]), en la literatura no hemos encontrado ningún tratamiento de generalización módulo teorías ecuacional. Nueva contribución en este ámbito es el desarrollo de una familia de algoritmos modulares para la E-generalización, donde la teoría $E$ es paramétrica: cualquier símbolo de función binario puede tener cualquier combinación de los axiomas de asociatividad, conmutatividad, e identidad (incluido el conjunto vacío de tales axiomas).

Para motivar mejor nuestro trabajo, recordemos primero el problema estándar de generalización. Sean $t_1$ y $t_2$ dos términos. Queremos encontrar un término $s$ que generalice los términos $t_1$ y $t_2$. En otras palabras, ambos, $t_1$ y $t_2$ deben ser instancias de $s$. Dicho término no es, en general, único. Por ejemplo, dados dos $t_1$ el término $f(f(a,a),b)$ y $f(f(b,b),a)$, entonces cualquier variable $x$ es trivialmente una generalización de los términos. Otra posible generalización es $f(x,y)$, también siendo $y$ un variable. El término $f(f(x,x),y)$ tiene la ventaja de ser el término más 'especifico' o *la generalización menos general (lgg)* (módulo renombramiento de variable).

En el caso en que los símbolos de función no satisfacen ningún axioma algebraico, el lgg es siempre único. Sin embargo, cuando queremos razonar *módulo* ciertos axiomas para los diferentes símbolos de función de nuestro problema, los lggs ya no tienen que ser únicos. Esto es análogo a los problemas de unificación ecuacional donde en general no hay un sólo mgu, sino un conjunto de mgus. Por ejemplo, consideremos que el anterior símbolo de función $f$ es asociativo y conmutativo. Entonces, el término $f(f(x,x),y)$ no es sólo el lgg de lo términos $f(f(a,a),b)$ y $f(f(b,b),a)$, porque existe otra generalización, $f(f(x,a),b)$, que es incomparable a la mencionada.

Al igual que en el caso de la unificación ecuacional [Siekmann, 1989], las cosas no son tan fáciles como para la generalización sintáctica, y el problema dual de calcular las $E$-generalizaciones menos generales es un problema difícil, sobre todo

en la gestión de la complejidad algorítmica del problema. La importancia de la generalización ecuacional ya se señaló por Pfenning en [Pfenning, 1991]: "It appears that the intuitiveness of generalizations can be significantly improved if anti–unification takes into account additional equations which come from the object theory under consideration. It is conceivable that there is an interesting theory of equational anti–unification to be discovered".

## 9.1. Estado del arte

A pesar de que la noción de generalización se remonta al trabajo de Plotkin [Plotkin, 1970], Reynolds [Reynolds, 1970], Pfenning [Pfenning, 1991], y Huet [Huet, 1976] y se ha estudiado en detalle por otros autores (véase por ejemplo el trabajo [Lassez et al., 1988]), no tenemos conocimiento de la existencia de ningún algoritmo de generalización ecuacional que calcule una generalización menos general para dos términos módulo cualquier combinación de los axiomas de asociatividad, conmutatividad e identidad. Plotkin [Plotkin, 1970] y Reynolds [Reynolds, 1970] dieron sendos algoritmos de generalización de forma imperativa, que esencialmente son el mismo. El algoritmo de generalización de Huet [Huet, 1976], formulado como un par de ecuaciones recursivas, no puede ser entendido como un cálculo automático, puesto que algunas asunciones (imprecisas) quedan implícitas en el tratamiento de variables, que se hacen de forma explicita en nuestra formulación. Pfenning [Pfenning, 1991] dió un algoritmo de generalización de orden superior para el cálculo de construcciones el cual no considera axiomas ecuacionales.

Una reconstrucción del algoritmo de Huet está dado en [Østvold, 2004] el cual no considera tipos. Finalmente, el algoritmo para la generalización en el calculo de construcciones de [Pfenning, 1991] no se puede utilizar para teorías order-sorted. En [Aït-Kaci and Sasaki, 2001], se da una definición de generación para *términos característicos* o "feature terms", una noción extendida de términos que también está basada en la tipología *order-sort*.

## 9.2. Contribuciones de la tesis

Las nuevas contribuciones originales que ahora se presentan y que configuran el cuerpo de la parte de cálculo de lgg son las siguientes:

**Un algoritmo modular de generalización ecuacional** (ver Capítulo 12) M. Alpuente, S. Escobar, J. Meseguer, P. Ojeda *A Modular Equational Generalization Algorithm*. 18th Int'l Symp. on Logic Based Program Synthesis and Tranformation. LOPSTR 2008, Valencia (Spain), Julio 2008.

En este trabajo, presentamos un algoritmo *modular* de generalización ecuacional, donde los símbolos de función pueden obedecer cualquier combinación de los axiomas de *asociatividad*, *conmutatividad* e *identidad*. En el marco de este trabajo, se ha desarrollado una herramienta web que implementa el algoritmos de generalización ecuacional y que esta públicamente disponible en

`http://www.dsic.upv.es/users/elp/toolsMaude/Lgg.html`.

**Generalización Order-Sorted** (ver Capítulo 13) M. Alpuente, S. Escobar, J. Meseguer, P. Ojeda. *Order-sorted Generalization*. 17th Int'l Workshop on Functional and (Constraint) Logic Programming, WFLP 2008. July 3-4, 2008 – Siena, Italy. Electronic Notes in Theoretical Computer Science, To appear 2008. Elsevier

En este trabajo, extendemos el algoritmo de generalización sintáctico a un algoritmo con tipos *order-sorted* mediante un sistema de reglas de inferencia y damos su prueba de corrección.

## 9.3. Estructura

El resto de esta parte del trabajo está organizado como sigue. En el Capítulo 10, se resumen algunos conceptos preliminares. En el Capítulo 11, recordaremos el algoritmo de generalización Huet [Huet, 1976], y proporcionaremos un novedoso algoritmo de generalización basado en reglas de inferencia que cómo veremos evita algunas asunciones implícitas del algoritmo de Huet. En el Capítulo 12, ampliamos el algoritmo sintáctico introducido en el Capítulo 11 proporcionando un algoritmo *modular* para una familia de teorías ecuacionales que incluye la

asociatividad, conmutatividad e identidad. En el Capítulo 13, generalizamos al caso *order-sorted* el algoritmo sin tipos o sintáctico presentado en el Capítulo 11 proporcionando un nuevo algoritmo capaz de generalizar términos en teorías *order-sorted*. Por último, en el Capítulo 14 describimos las conclusiones de esta parte de la tesis y trabajos futuros.

# 10

# Preliminares

En esta sección, recordamos algunas definiciones preliminares junto con la terminología necesaria para el entendimiento de esta parte del trabajo.

Seguimos la notación y terminología clásica de [TeReSe, 2003] para reescritura de términos y [Meseguer, 1992; Meseguer, 1997] para lógica de reescritura y notaciones order-sorted. Asumimos una *signatura order-sorted* $\Sigma$ con un poset finito de sotrs $(\mathsf{S}, \leq)$ y un número finito de símbolos de funciones. Además asumimos que: (i) cada componente conectada en el orden del poset tiene un top sort, y para cada $\mathsf{s} \in \mathsf{S}$ denotamos por $[\mathsf{s}]$ el top sort en la componente de $\mathsf{s}$; y (ii) para cada declaración de operador $f : \mathsf{s_1} \times \ldots \times \mathsf{s_n} \to \mathsf{s}$ en $\Sigma$, hay también una declaración $f : [\mathsf{s_1}] \times \ldots \times [\mathsf{s_n}] \to [\mathsf{s}]$.

A lo largo de esta parte de la tesis, asumimos que $\Sigma$ no tiene ningún *operador sobrecargado*, por ejemplo, cualquier declaración de dos operadores para el mismo símbolo $f$ con el mismo número de argumentos, $f : \mathsf{s_1} \times \ldots \times \mathsf{s_n} \to \mathsf{s}$ y $f : \mathsf{s'_1} \times \ldots \times \mathsf{s'_n} \to \mathsf{s'}$, debe necesariamente tener $[\mathsf{s_1}] = [\mathsf{s'_1}], \ldots, [\mathsf{s_n}] = [\mathsf{s'_n}], [\mathsf{s}] = [\mathsf{s'}]$. Asumimos una familia $\mathsf{S}$-sorted $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ de conjuntos de variables disjuntas con cada $\mathcal{X}_\mathsf{s}$ contablemente infinita.

Un variable *fresca* es una variable que no aparece en ninguna parte.

$\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ es el conjunto de términos del sort $\mathsf{s}$, y $\mathcal{T}_{\Sigma,\mathsf{s}}$ es el conjunto de términos ground del conjunto $\mathsf{s}$. Escribimos $\mathcal{T}(\Sigma, \mathcal{X})$ y $\mathcal{T}(\Sigma)$ para la correspondiente álgebra de términos. Asumimos que $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ para cada sort $\mathsf{s}$.

Para un término $t$, escribimos $Var(t)$ para el conjunto de todas las variables *preserva los sorts* en $t$. Las posiciones de los términos son representadas como cadenas de números naturales y son dotadas con el prefijo de ordenación $\leq$ sobre cadenas. El conjunto de posiciones de un término $t$ es escrito como $Pos(t)$, y el

conjunto de posiciones no variables se escribe $Pos_\Sigma(t)$. La posición raíz de un término es $\Lambda$.

El subtérmino de $t$ en la posición $p$ es $t|_p$ y $t[u]_p$ es el término $t$ donde $t|_p$ es remplazado por $u$. Por $root(t)$ denotamos el símbolo que aparece en la posición raíz de $t$.

Una *sustitución* $\sigma$ es un remplazamiento de un subconjunto finito de $\mathcal{X}$, escritas $Dom(\sigma)$, hasta $\mathcal{T}(\Sigma, \mathcal{X})$. El conjunto de variables introducidas por $\sigma$ es $Ran(\sigma)$. La sustitución identidad es *id*. Las sustituciones son homomórficamente extentidas a $\mathcal{T}(\Sigma, \mathcal{X})$. La aplicación de una sustitución $\sigma$ a un término $t$ es denotado por $t\sigma$. La restricción de $\sigma$ para un conjunto de variables $V$ es $\sigma|_V$. Una composición de dos sustituciones se denota por la yuxtaposición, por ejemplo, $\sigma\sigma'$. Llamamos una sustitución $\sigma$ un *renombramiento* si hay otra sustitución $\sigma^{-1}$ tal que $\sigma\sigma^{-1}|_{Dom(\sigma)} = id$. Las sustituciones preservan los sorts de las variables, por ejemplo, para cualquier sustitución $\sigma$, si $x \in \mathcal{X}_\mathsf{s}$, entonces $x\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$.

Una *regla de reescritura* es un par orientado $l \to r$, donde $l \notin \mathcal{X}$, $Var(r) \subseteq Var(l)$, y $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ para algún sort $\mathsf{s} \in \mathsf{S}$. Asumimos que las reglas son *sort-decrecientes*, por ejemplo, para cada regla $t \to t'$, $\mathsf{s} \in \mathsf{S}$, y la sustitución $\sigma$, $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ implica $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. Una *teoría de reescritura* order-sorted *(incondicional)* es una tupla $\mathcal{R} = (\Sigma, R)$ con $\Sigma$ una signatura order-sorted, y $R$ un conjunto de reglas de reescritura. La relación de reescritura sobre $\mathcal{T}(\Sigma, \mathcal{X})$, escrito $t \xrightarrow{p}_R t'$ (o $t \to_R t'$) se cumple si y solo si existe $p \in Pos_\Sigma(t)$, $l \to r \in R$ y una sustitución $\sigma$, tal que $t|_p = l\sigma$, y $t' = t[r\sigma]_p$.

Una $\Sigma$-*ecuación* es un par no orientado $t = t'$, donde $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ para algún sort $\mathsf{s} \in \mathsf{S}$. Una *teoría ecuacional* $(\Sigma, E)$ es un conjunto de $\Sigma$-ecuaciones. Una teoría ecuacional $(\Sigma, E)$ es *regular* si para cada $t = t' \in E$, tenemos $Var(t) = Var(t')$, y si para cada sustitución $\sigma$, tenemos $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ si y solo si $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$, y todas las variables en $Var(t)$ tienen un sort mínimo o más específico (sort *top*). Dados $\Sigma$ y un conjunto $E$ de $\Sigma$-ecuaciones, la lógica ecuacional order-sorted induce a la relación de congruencia $=_E$ sobre términos $t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$ (ver [Meseguer, 1997]), por ejemplo, $t =_E t'$ si y solo si $t \to^*_{\overleftrightarrow{E}} t'$ donde $\overleftrightarrow{E} = \{l \to r,\ r \to l \mid l = r \in E\}$.

Consideramos teorías ecuacionales que contienen las siguientes ecuaciones (también llamadas axiomas) para un símbolo de función $f$: asociatividad,

$f(X, f(Y, Z)) = f(f(X, Y), Z)$, conmutatividad, $f(X, Y) = f(Y, X)$, e identidad, $f(X, 0) = X$.

Escribimos el sort asociado a una variable explícitamente con dos puntos y el sort, por ejemplo $x{:}\mathsf{Nat}$. Asumimos *pre-regularidad* de la signatura $\Sigma$, asegurando que cada término $t$ tiene un único sort mínimo, denotado por $LS(t)$.

Por lo cual, el sort top en la componente conectada de $LS(t)$ se denota por $[LS(t)]$. Puesto el poset $(\mathsf{S}, \leq)$ es finito y cada componente conectada tiene un sort top, dados culquier par de sorts $\mathsf{s}$ y $\mathsf{s}'$ en la misma componente conectada, el conjunto de "least upper bound sorts" de $\mathsf{s}$ y $\mathsf{s}'$, aunque no necesariamente un sólo conjunto, siempre existe y es denotado por $LUBS(\mathsf{s}, \mathsf{s}')$.

# 11

# LGG Sintáctico

Plotkin [Plotkin, 1970] y Reynolds [Reynolds, 1970] formularon un algoritmo al estilo imperativo de generalización, que son ambos esencialmente los mismos. Huet dio un nuevo algoritmo de generalización [Huet, 1976] formulados como un par de ecuaciones recursivas.

En esta sección revisitamos la generalización estándar formalizando un nuevo sistema original que nos será útil en nuestras siguientes extensiones de este algoritmo para el caso ecuacional 12 y el caso order-sorted 13.

Sea $\leq$ la instanciación estándar de cuasi - ordenación sobre términos dados por la relación de ser "más general", por ejemplo $t$ es más general que $s$ (i.e. $s$ es una instancia de $t$), escrito $t \leq s$, si y solo si existe $\theta$ tal que $t\theta = s$. La unificación más general de un conjunto $M$ es el menor límite superior (la instancia más general) de $M$ bajo $\leq$. La generalización corresponde al mayor límite inferior. Dado un conjunto no vacío $M$ de términos, el término $w$ es una generalización de $M$ si, para todo $s \in M$, $w \leq s$. Un término $w$ es la generalizaicón menos general de $M$ if $w$ es una generalización de $M$ y, para cada otra generalización $u$ de $M$, $u \leq w$.

El algoritmo de generalización no determinista $\lambda$ de Huet [Huet, 1976] (tratado en detalle en [Lassez et al., 1988]) es como sigue. Sea $\Phi$ una biyección entre $\mathcal{T}(\Sigma, \mathcal{X}) \times \mathcal{T}(\Sigma, \mathcal{X})$ y un conjunto de variables $V$. La función recursiva $\lambda$ sobre $\mathcal{T}(\Sigma, \mathcal{X}) \times \mathcal{T}(\Sigma, \mathcal{X})$ que calcula el lgg de dos términos está dada por:

- $\lambda(f(s_1, \ldots, s_m), f(t_1, \ldots, t_m)) = f(\lambda(s_1, t_1), \ldots, \lambda(s_m, t_m))$, para $f \in \Sigma$
- $\lambda(s, t) = \Phi(s, t)$, en otro caso.

La función global $\Phi$ es la parte central de este algoritmo que se utiliza para garantizar que las mismas desavenencias entre términos son remplazadas por la misma

variable en ambos términos. Por otro lado, existen algunas asunciones implícitas (imprecisas) en el algoritmo sobre le tratamiento de variables que se esconden en la función global $\Phi$. De hecho, no existe una condición explicita sobre $\Phi$ para el manejo de variables, es decir, que las variables de las condiciones iniciales podrían no guardarse en $\Phi$. Por ejemplo, consideremos la siguiente función $\Phi$ de [Lassez et al., 1988] la cual no satisface la condición de buen comportamiento: $\Phi(x, a) = x$ y $\Phi(b, a) = y$. Esta función devuelve $\lambda(f(x, f(x, f(a, b))), f(a, f(a, f(a, a))) = f(x, f(x, f(a, y)))$, lo cual es correcto. Por otro lado, si añadimos $\Phi(x, b) = x$, esta función produce un resultado erróneo en $\lambda(f(x, x), f(a, b)) = f(x, x)$. La solución para reparar este error es remplazar todas las ocurrencias de una variable $x$ en el término original por una nueva constante $c_x$ antes de utilizarla en el algoritmo y, después de ejecutar el algoritmo, reemplazar $c_x$ por $x$ en el resultado.

Ahora, proporcionamos un conjunto *novedoso* de reglas de inferencia para calcular el lgg de dos términos, evitando asunciones implícitas, y oscuras ya que utilizamos un repositorio o *store* donde guardamos las generalizaciones de subproblemas ya resueltos. Este algoritmo también se puede utilizar (gracias a a la asociatividad y conmutatividad del lgg) para calcular, de una forma obvia, el lgg de un conjunto arbitrario de términos mediante el cálculo sucesivo del lgg de dos términos del conjunto.

En nuestra reformulación, representamos un problema de generalización entre los términos $s$ y $t$ como una restricción o *constraint* $s \overset{x}{\triangleq} t$, donde $x$ es una variable fresca que permanece para una generalización (más general) de $s$ y $t$. Por medio de esta representación, cualquier generalización $w$ de $s$ y $t$ es dada por una sustitución $\theta$ tal que $x\theta = w$.

Calculamos la generalización menos general de $s$ y $t$ por medio de un sistema de transición $(Conf, \rightarrow)$ [Plotkin, 2004] donde *Conf* es un conjunto de *configuraciones* y la relación de transición $\rightarrow$ es dada por un conjunto de reglas de inferencia.

Además de la *componente constraint*, por ejemplo, un conjunto de constraints de la forma $t_i \overset{x_i}{\triangleq} t_{i'}$, y el *componente sustitución*, por ejemplo, la sustitución parcial calculada hasta ahora, las configuraciones también incluyen un componente extra, llamado *almacén* o *store*. Este store[1] "juega" el rol de la función $\Phi$ del algoritmo

[1]Nuestra noción de store es comparable a la del conjunto *history* de *Hist* en [Aït-Kaci and

de generalización de Huet, con la diferencia que nuestro nuestros *store* son locales al sistema de configuraciones, mientras que $\Phi$ puede ser entenderse como un repositorio global. Podemos ver que la no-globalidad del *store* es la clave para calcular un conjunto completo y mínimo de soluciones.

**Definición 11.0.1** *Una configuración, escrita como* $\langle C \mid S \mid \theta \rangle$*, consiste de tres componentes:*

- *el* componente constraint C*, una conjunción* $s_1 \overset{x_1}{\triangleq} t_1 \wedge \ldots \wedge s_n \overset{x_n}{\triangleq} t_n$ *que representa* el conjunto de constraints por resolver
- *el* componente store S*, que almacena el* conjunto de constraints ya resueltas*, y*
- *el* componente sustitución $\theta$*, que contiene los remplazamientos para alguna de las variables* $x_1, \ldots, x_n$ *presentes en las constraints* $s_i \overset{x_i}{\triangleq} t_i$ *de* $C$ *y* $S$*.*

Empezando con la configuración inicial $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id \rangle$, vamos transformando las configuraciones hasta que alcanzamos una configuración final $\langle \emptyset \mid S \mid \theta \rangle$. Entonces, el lgg de $t$ y $t'$ es simplemente $x\theta$. Como veremos, $\theta$ es único módulo renombramiento de variables.

La relación de transición $\rightarrow$ está dada por el conjunto de relaciones más pequeño que satisfacen las reglas de la Figura 11.1. En esta parte de la tesis, las variables de los términos $t$ y $s$ en un problema de generalización $t \overset{x}{\triangleq} s$ se consideran como constantes, puesto que nunca son instanciadas. El significado de las reglas es como sigue:

- La regla **Decompose** es la decomposición sintáctica, la cual genera nuevas *constraints* a ser solucionadas.
- La regla **Recover** comprueba si una constraint $t \overset{x}{\triangleq} s \in C$ con $root(t) \not\equiv root(s)$ ya ha sido solucionada. Por ejemplo, si ya hay una $t \overset{y}{\triangleq} s \in S$ para el mismo *par de conflicto* $(t, s)$, con variable $y$. Esto se necesita cuando en la entrada de los términos del problema de generalización contiene el mismo par de conflicto más de una vez, por ejemplo, el lgg de $f(a, a, a)$ y $f(b, b, a)$ es $f(y, y, a)$.

Sasaki, 2001], aunque se nos ocurrió la idea del store de forma independiente.

**Decompose**
$$\frac{f \in (\Sigma \cup \mathcal{X})}{\langle f(t_1,\ldots,t_n) \overset{x}{\triangleq} f(t'_1,\ldots,t'_n) \wedge C \mid S \mid \theta\rangle \rightarrow \langle t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge C \mid S \mid \theta\sigma\rangle}$$
donde $\sigma = \{x \mapsto f(x_1,\ldots,x_n)\}$, $x_1,\ldots,x_n$ son variables frescas , y $n \geq 0$

**Solve**
$$\frac{root(t) \not\equiv root(t') \wedge \nexists y : t \overset{y}{\triangleq} t' \in S}{\langle t \overset{x}{\triangleq} t' \wedge C \mid S \mid \theta\rangle \rightarrow \langle C \mid S \wedge t \overset{x}{\triangleq} t' \mid \theta\rangle}$$

**Recover**
$$\frac{root(t) \not\equiv root(t')}{\langle t \overset{x}{\triangleq} t' \wedge C \mid S \wedge t \overset{y}{\triangleq} t' \mid \theta\rangle \rightarrow \langle C \mid S \wedge t \overset{y}{\triangleq} t' \mid \theta\sigma\rangle}$$
donde $\sigma = \{x \mapsto y\}$

Figura 11.1: Reglas para la generalización menos general.

- La regla **Solve** comprueba si una constraint $t \overset{x}{\triangleq} s \in C$ con $root(t) \not\equiv root(s)$, no ha sido previamente resuelta. Si no está solucionada, la constraint $t \overset{x}{\triangleq} s$ se añade al store $S$.

Note que las reglas de inferencia de la Figura 11.1 no son deterministas (es decir, la elección de la constraint del conjunto $C$). No obstante, son confluentes módulo renombramiento de variables (es decir, la elección de la transición es irrelevante para el cálculo de las configuraciones finales). Esto justifica que la generalización menos general de dos términos es única módulo renombramiento de variable [Lassez et al., 1988].

**Ejemplo 11.0.2** *Sean $t = f(g(a), g(y), a)$ y $s = f(g(b), g(y), b)$ dos términos. Aplicamos las reglas de inferencia de la figura 11.1 y la sustitución obtenida por el algoritmo lgg es $\theta = \{x \mapsto f(g(x_4), g(y), x_4), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y, x_3 \mapsto x_4\}$. Por tanto, el lgg es $x\theta = f(g(x_4), g(y), x_4)$. Note que la variable $x_4$ está repetida, para asegurar la generalización menos general. Puede observar la traza de ejecución de las reglas en la Figura 11.2.*

Demostrar la terminación y confluencia (con renombramiento de variables) del sistema de transición $(Conf, \rightarrow)$ es sencillo.

**Teorema 11.0.3** *Cada derivación de una configuración inicial $\langle t \overset{x}{\triangleq} s \mid \emptyset \mid id\rangle$ termina.*

$$
\begin{array}{c}
lgg(f(g(a),g(y),a),f(g(b),g(y),b)) \\
\downarrow \quad \text{Configuración Inicial} \\
\langle f(g(a),g(y),a) \overset{x}{\triangleq} f(g(b),g(y),b) \mid \emptyset \mid id\rangle \\
\downarrow \quad \text{Decompose} \\
\langle g(a) \overset{x_1}{\triangleq} g(b) \wedge g(y) \overset{x_2}{\triangleq} g(y) \wedge a \overset{x_3}{\triangleq} b \mid \emptyset \mid \{x \mapsto f(x_1,x_2,x_3)\}\rangle \\
\downarrow \quad \text{Decompose} \\
\langle a \overset{x_4}{\triangleq} b \wedge g(y) \overset{x_2}{\triangleq} g(y) \wedge a \overset{x_3}{\triangleq} b \mid \emptyset \mid \{x \mapsto f(g(x_4),x_2,x_3), x_1 \mapsto g(x_4)\}\rangle \\
\downarrow \quad \text{Solve} \\
\langle g(y) \overset{x_2}{\triangleq} g(y) \wedge a \overset{x_3}{\triangleq} b \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),x_2,x_3), x_1 \mapsto g(x_4)\}\rangle \\
\downarrow \quad \text{Decompose} \\
\langle y \overset{x_5}{\triangleq} y \wedge a \overset{x_3}{\triangleq} b \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),g(x_5),x_3), x_1 \mapsto g(x_4), x_2 \mapsto g(x_5)\}\rangle \\
\downarrow \quad \text{Decompose} \\
\langle a \overset{x_3}{\triangleq} b \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),g(y),x_3), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y\}\rangle \\
\downarrow \quad \text{Recover} \\
\langle \emptyset \mid a \overset{x_4}{\triangleq} b \mid \{x \mapsto f(g(x_4),g(y),x_4), x_1 \mapsto g(x_4), x_2 \mapsto g(y), x_5 \mapsto y, x_3 \mapsto x_4\}\rangle
\end{array}
$$

Figura 11.2: Traza del cálculo del lgg de los términos $f(g(a),g(y),a)$ y $f(g(b),g(y),b)$.

**Proof.** Sea $|u|$ el número de ocurrencias de un símbolo en un término $u$. Puesto que el mínimo de $|t|$ y $|s|$ es un límite superior para el número de veces que las reglas de inferencia se pueden aplicar, entonces la derivación termina. □

Antes de probar la corrección y completitud de las reglas de inferencia anteriores, necesitamos algunos conceptos auxiliares para una posición conflictiva y pares conflictivos, y la cuál formalizamos mediante tres lemas auxiliares. Dados los términos $t$ y $t'$, una posición $p \in \mathcal{P}os(t) \cap \mathcal{P}os(t')$ se llama *una posición conflictiva de $t$ y $t'$* si $root(t|_p) \not\equiv root(t'|_p)$ y para todo $q < p$, $root(t|_q) \equiv root(t'|_q)$. El par $(t|_p, t'|_p)$ se llama un *par conflictivo de $t$ y $t'$*. También, note que dada una constraint $t \overset{x}{\triangleq} t'$, $x$ es siempre una generalización (más general) de $t$ y $t'$.

**Lema 11.0.4** *Dados los términos $t$ y $t'$ y una variable fresca $x$ tal que $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$, una constraint $u \overset{y}{\triangleq} v$ está en $S$ si y solo si existe una posición conflictiva $p$ de $t$ y $t'$ tal que $t|_p = u$ y $t'|_p = v$.*

**Lema 11.0.5** *Dados los términos $t$ y $t'$ y una variable fresca $x$ tal que $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle C \mid S \mid \theta\rangle$, entonces $x\theta$ es una generalización de $t$ y $t'$.*

**Lema 11.0.6** *Dados los términos $t$ y $t'$ y una variable fresca $x$ tal que $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$, entonces $\{y \in \mathcal{X} \mid \exists u \overset{y}{\triangleq} v \in S\} \subseteq Ran(\theta)$, y $Ran(\theta) = Var(x\theta)$.*

La completitud y corrección se prueban como sigue.

**Teorema 11.0.7** *Dados los términos $t$ y $t'$ y una variable fresca $x$, $u$ es el lgg de $t$ y $t'$ si y solo si $\langle t \overset{x}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ y hay un renombramiento $\rho$ tal que $u\rho = x\theta$.*

**Proof.** Contamos con la ya conocida existencia y unicidad del lgg de $t$ y $t'$ y razonamos por contradicción. Por el Lema 11.0.5, $x\theta$ es una generalización de $t$ y $t'$. Si $x\theta$ no es el lgg de $t$ y $t'$, entonces hay un término $u$ el cual es el lgg de $t$ y $t'$ y una sustitución $\rho$ que no es un renombramiento de variables y $x\theta\rho = u$. Dado que, por el Lema 11.0.6, $Ran(\theta) = Var(x\theta)$, podemos elegir siempre $\rho$ con $Dom(\rho) = Var(x\theta)$. Si $\rho$ no es un renombramiento de variable, o bien:

1. Hay variables $y, y' \in Var(x\theta)$ y una variable $z$ tal que $y\rho = y'\rho = z$, o
2. Hay una variable $y \in Var(x\theta)$ y un término que no es una variable $v$ tal que $y\rho = v$.

En el caso (1), hay dos posiciones conflictivas $p, p'$ para $t$ y $t'$ tal que $u|_p = z = u|_{p'}$ y $x\theta|_p = y$ y $x\theta|_{p'} = y'$. En particular, esto significa que $t|_p = t|_{p'}$ y $t'|_p = t'|_{p'}$. Pero esto es imposible por los Lemas 11.0.4 y 11.0.6. En el caso (2), hay una posición $p$ tal que $x\theta|_p = y$ y $p$ no es ni una posición conflictiva de $t$, y $t'$ no es posición conflictiva de $t$ y $t'$. Pero esto es imposible por los Lemas 11.0.4 y 11.0.6. □

# 12

# Algoritmo modular de Generalización Ecuacional

En este apartado se estudia en detalle un algoritmo *modular* para una familia de teorías ecuacionales más comunes, como pueden ser todas las teorías de $f \in \Sigma$ tal que cada símbolo de función binario $f \in \Sigma$ puede tener cualquier combinación de los siguientes axiomas ecuacionales: (i) *asociatividad* $(A_f)$ $f(x, f(y, z)) = f(f(x, y), z)$; (ii) *conmutatividad* $(C_f)$ $f(x.y) = f(y, x)$, (iii) *identidad* $(U_f)$ para un símbolo constante $e$, por ejemplo, $f(e, x) = x$ y $f(e, x) = x$. En particular, $f$ podría no satisfacer ninguno de esos axiomas. Cuando esto suceda para todos los símbolos binarios $f \in \Sigma$ obtenemos el algoritmo estándar de generalización como un caso especial.

Las principales contribuciones en este apartado son:

- Un algoritmo de generalización modular especificado mediante un conjunto de reglas de inferencia, donde diferentes símbolos de funciones satisfacen los axiomas asociatividad y/o conmutatividad y/o identidad, teniendo los axiomas diferentes reglas de inferencia.

- Una prueba de corrección de las reglas de inferencia.

El algoritmo es *modular* en el sentido preciso que la combinación de los diferentes axiomas ecuacionales para los diferentes símbolos de función es automático: las reglas de inferencia pueden ser aplicadas a los problemas de generalización sin necesidad de realizar ningún cambio o adaptación. Esto se realiza de forma similar en la combinación. Esto es parecido, aunque mucho más simple y más fácil, que

los métodos modulares para la combinación de algoritmos de $E$-unificación (por ejemplo, [Baader and Snyder, 1999]).

Como ya hemos mencionado, nuestro algoritmo de $E$-generalización debería ser de gran interés para desarrolladores de lenguajes basados en reglas, demostradores de teoremas y programas ecuacionales de razonamiento, así como herramientas de manipulación de programas como por ejemplo analizadores de programas y evaluadores parciales, para lenguajes declarativos y sistemas de razonamiento que soporten eficientemente axiomas ecuacionales tales como asociatividad, conmutatividad e identidad. Por ejemplo, esto incluye algunos demostradores de teoremas, y una variedad de lenguajes basados en reglas como ASF+SDF, OBJ, CafeOBJ, Elan y Maude.

Cuando tenemos una teoría ecuacional $E$, debemos revisar la noción de la generalización menos general, ya que pueden existir $E$-generalizaciones de los términos sin que exista un único lgg. Al igual que el caso dual de $E$-unificación, tenemos que hablar de un *conjunto* de $E$-lggs.

**Ejemplo 12.0.8** *Consideremos los términos $t = f(a, a, b)$ y $s = f(b, b, a)$ donde $f$ es asociativa y conmutativa, y $a$ y $b$ son constantes. Los términos $u = f(x, x, y)$ y $u' = f(x, a, b)$ son generalizaciones de $t$ y $s$ pero no son comparables, es decir, ninguno es una instancia del otro módulo los axiomas AC de $f$.*

Nuestro algoritmo de de generalización ecuacional módulo una teoría ecuacional arbitraria $E$, denotado por $gen_E(s, t)$, es un algoritmo que es capaz de calcular un conjunto completo $\Theta$ de $E$-generalizaciones para todos los posibles elementos en las clases de equivalencias de los términos $s$ y $t$. Sin embargo, las generalizaciones menos generales tienen que ser obtenidas por un post-proceso de filtrado. Esto es posible siempre porque, bajo condiciones apropiadas que exigimos en la teoría ecuacional $E$ y que se describen a continuación, el conjunto $\Theta$ de $E$-generalizaciones de $s$ y $t$ también es finito.

Esto es diferente del problema dual de $E$-unificación, donde el conjunto de $E$-unificadores de dos términos $s$ y $t$ no es necesariamente finito, por lo que no existe en general un completo, finito y minimal conjunto de $E$–unificadores más generales .

Dados dos términos $t$ y $s$ y una teoría ecuacional $E$, escribimos $t <_E s$ si y solo si $\exists\theta : t\theta =_E s$ y $\theta$ no es un renombramiento.

**Definición 12.0.9** *Sea $t$ y $s$ dos términos y $E$ una teoría ecuacional. Un conjunto completo de generalizaciones de $t$ y $s$ módulo $E$, denotado por $gen_E(t,s)$, se define como sigue: $gen_E(t,s) = \{v \mid \exists u, u' \text{ tal que } t =_E u \wedge t' =_E u' \wedge lgg(u,u') = v\}$ El conjunto de generalizaciones menos generales de $t$ y $s$ módulo $E$ se define como sigue:*

$$lgg_E(t,t') = minimal_{<_E}(gen_E(t,t'))$$

*donde $minimal_{<_E}(S) = \{s \in S \mid \nexists s' \in S : s' <_E s\}$. Los lggs ecuacionales son equivalentes módulo renombramiento, y por lo tanto, eliminamos de $lgg_E(t,t')$ las versiones renombradas de los términos.*

Note que las definiciones previas requieren el matching módulo $E$ para calcular el conjunto de generalizaciones menos generales. El matching es decidible para la asociatividad, conmutatividad, idententidad, y sus combinaciones; además existen algoritmos eficientes.

Enumerando todos los elementos en la clase de equivalencia de cada término $s$ y $t$, este algoritmo termina cuando la teoría ecuacional $E$ es *finita*, es decir, las clases de equivalencia contienen un número finito de términos, tales como aquellas teorías que contienen los axiomas de asociatividad y conmutatividad. Por otro lado, este algoritmo es muy ineficiente. Además, las teorías ecuacionales que contienen axiomas tales como identidad no son finitas y el algoritmo anterior no funciona. El siguiente resultado es inmediato.

**Teorema 12.0.10** *Dados los términos $t$ y $s$, una teoría ecuacional $E$, $u \in lgg_E(t,s)$ es un conjunto correcto y completo de lggs de $t$ y $s$ (con renombramiento).*

Dada una teoría ecuacional finita $E$, y dos términos $t$ y $s$, siempre podemos enumerar recursivamente este conjunto, el cuál es por contrucción un conjunto completo de generalizaciones de $t$ y $s$. Por esto, solo necesitamos enumerar recursivamente todos los pares de términos $(u,u')$ con $t =_E u$ y $s =_E u'$. Por tanto, este conjunto $gen_E(t,s)$ puede ser posiblemente infinito. Por otro lado, si la teoría $E$ tiene además la propiedad de que cada clase de la $E$-equivalencia es *finita* y puede ser generada eficazmente, entonces el proceso anterior es un *algoritmo* terminante, generando un conjunto finito de generalizaciones de $t$ y $s$.

En cualquier caso, para cualquier teoría finita $E$, podemos caracterizar siempre matemáticamente un *conjunto completo minimal* de $E$-generalizaciones, es decir, un conjunto $lgg_E(t, s)$ definido como sigue.

**Definición 12.0.11** *Sean $t$ y $s$ términos y $E$ una teoría ecuacional. Un conjunto completo de generalizaciones de $t$ y $s$ módulo $E$, denotado por $gen_E(t, s)$, se define como sigue: $gen_E(t, s) = \{v \mid \exists u, u' \text{ s.t. } t =_E u \wedge s =_E u' \wedge lgg(u, u') = v\}$.*
*El conjunto de generalizaciones menos generales de $t$ y $s$ módulo $E$ se define como sigue:*

$$lgg_E(t, s) = minimales_{<_E}(gen_E(t, s))$$

*donde $minimales_{<_E}(S) = \{s \in S \mid \nexists s' \in S : s' <_E s\}$. Los lggs son equivalentes módulo renombramiento y, por lo tanto, eliminamos de $lgg_E(t, t')$ versiones renombradas de los términos.*

El siguiente resultado es inmediato.

**Teorema 12.0.12** *Dados los términos $t$ y $s$ en una teoría ecuacional $E$, $lgg_E(t, s)$ es un conjunto minimal, correcto, y completo de lggs módulo $E$ de $t$ y $s$ (módulo renombramiento).*

Sin embargo, note que en general la relación $t <_E t'$ es *indecidible*, ya que el anterior conjunto, aunque esté bien definido al nivel matemático, no puede ser calculado. Sin embargo, cuando (i) cada clase $E$-equivalencia es *finita* y puede ser generada; y (ii) hay un algoritmo de $E$-matching, entonces tenemos un algoritmo para calcular el $lgg_E(t, s)$, puesto que la relación $t <_E t'$ es preciasamente la relación de $E$-matching.

En resumen, cuando $E$ es finita y satisface las condiciones (i) y (ii), las definiciones anteriores nos dan un procedimiento correcto, aunque horriblemente ineficiente, para calcular un conjunto finito, minimal y completo de generalizaciones menos generales $lgg_E(t, s)$. Este sencillo algoritmo puede ser utilizado cuando $E$ consiste en los axiomas de asociatividad y/o conmutatividad para los símbolos de función, porque tales teorías (un caso especial de nuestras familias de teorías paramétricas) satisfacen las condiciones (i)–(ii). Sin embargo, cuando añadimos el axioma de identidad, las clases $E$-equivalencia se convierten en infinitas, de modo que el anterior esquema no nos da un algoritmo de lgg módulo $E$.

En las siguientes secciones, proporcionamos un algoritmo modular, minimal, terminante, correcto y completo para teorías ecuacionales que contienen diferentes axiomas como asociatividad, conmutatividad e identidad (y sus combinaciones). El conjunto $lgg_E(t, s)$ de $E$-generalizaciones menos generales se calcula en dos fases: (i) primero calculamos mediante las reglas de inferencia siguientes, un conjunto completo de $E$-generalizaciones, y entonces (ii) Filtramos el conjunto de $E$-generalizaciones para obtener $lgg_E(t, s)$ utilizando el hecho que para todas las teorías $E$ en la familia de las teorías paramétricas que consideramos, hay un algoritmo de matching módulo $E$. Consideramos que un símbolo de función $f$ en la signatura $\Sigma$ cumple un subconjunto de axiomas $ax(f) \subseteq \{A_f, C_f, U_f\}$. En particular, $f$ puede no satisfacer ningún axioma, $ax(f) = \emptyset$.

Proporcionamos nuestras reglas de inferencia para la generalización ecuacional de una manera "paso a paso". Primero, $ax(f) = \emptyset$, luego, $ax(f) = \{C_f\}$, luego, $ax(f) = \{A_f\}$, luego, $ax(f) = \{A_f, C_f\}$, y finalmente, $\{U_f\} \in ax(f)$. Técnicamente, las variables de los términos originales se manejan en nuestras reglas como constantes. Por eso para cualquier variable $x \in X$, consideramos $ax(x) = \emptyset$.

## 12.1. Reglas básicas para la $E$–generalización

Empezamos con un conjunto básico de reglas que son la versión ecuacional de la generalziación sintáctica de las reglas del Capítulo 11. La regla $Decompose_E$ se aplica a los símbolos de función que no poseen ningún axioma, $ax(f) = \emptyset$. Seguidamente, proporcionamos reglas específicas para la decomposición de *constraints* de los términos que tienen como raíz símbolos que poseen axiomas ecuacionales, tales como ACU y sus combinaciones.

Con respecto a las reglas $Solve_E$ y $Recover_E$, la principal diferencia con respecto a las reglas correspondientes de generalización sintáctica dadas en el Capítulo 11 es en el hecho de que la comprobación de la existencia de las *constraints* del *store* se realiza módulo $E$: en las siguiente reglas, escribimos $t \overset{y}{\triangleq} t' \in^E S$ para expresar que existe $s \overset{y}{\triangleq} s' \in S$ tal que $t =_E s$ y $t' =_E s'$.

Finalmente, en lo que respecta a la regla $Solve_E$, note que esta regla no puede ser aplicada a cualquier constraint $t \overset{x}{\triangleq} s$, se aplica sólo a aquellas reglas que o bien $t$ o $s$ tienen como raíz un símbolo de función $f$ con $U_f \in ax(f)$. Para los símbolos

**Decompose**$_E$
$$\frac{f \in (\Sigma \cup \mathcal{X}) \wedge ax(f) = \emptyset}{\langle f(t_1,\ldots,t_n) \overset{x}{\triangleq} f(t'_1,\ldots,t'_n) \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle t_1 \overset{x_1}{\triangleq} t'_1 \wedge \ldots \wedge t_n \overset{x_n}{\triangleq} t'_n \wedge CT \mid S \mid \theta\sigma\rangle}$$
donde $\sigma = \{x \mapsto f(x_1,\ldots,x_n)\}$, $x_1,\ldots,x_n$ son variables frescas, y $n \geq 0$

**Solve**$_E$
$$\frac{f = root(t) \wedge g = root(t') \wedge f \not\equiv g \wedge U_f \notin ax(f) \wedge U_g \notin ax(g) \wedge \nexists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle CT \mid S \wedge t \overset{x}{\triangleq} t' \mid \theta\rangle}$$

**Recover**$_E$
$$\frac{root(t) \not\equiv root(t') \wedge \exists y : t \overset{y}{\triangleq} t' \in^E S}{\langle t \overset{x}{\triangleq} t' \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle CT \mid S \mid \theta\sigma\rangle}$$
donde $\sigma = \{x \mapsto y\}$

Figura 12.1: Reglas básicas para la $E$–generalización menos general.

de función con un elemento identidad, en la Sección 12.5 daremos una regla especial $Expand_U$ que nos permite resolver una *constraint* $f(t_1,t_2) \overset{x}{\triangleq} s$, tal que $root(s) \not\equiv f$, con una generalización $f(y,z)$ más específica que $x$, introduciendo primero la constraint$f(t_1,t_2) \overset{x}{\triangleq} f(s,e)$.

## 12.2. Generalización menos general módulo $C$

En esta sección extendemos el conjunto básico de reglas de generalización ecuacional añadiendo una regla específica de inferencia $Decompose_C$, mostradas en la Figura 12.2, para operar con símbolos de función que obedecen el axioma de conmutatividad.

Esta regla de inferencia reemplaza la regla de inferencia sintáctica de decomposición (*Decompose*) para el caso de un símbolo binario conmutativo $f$, es decir, se consideran los cuatro posibles reorganizaciones de los términos $f(t_1,t_2)$ y $f(s_1,s_2)$.

Nótese que esta regla no es determinista, por lo tanto, las cuatro combinaciones deben ser exploradas.

**Ejemplo 12.2.1** *Sean $t = f(a,b)$ y $s = f(b,a)$ dos términos donde $f$ es conmutativa, es decir, $C_f \in ax(f)$. Aplicando las reglas $Solve_E$, $Recover_E$, y $Decompose_C$ anteriores, terminamos en una configuración final $\langle\emptyset \mid S \mid \theta\rangle$, donde $\theta = \{x \mapsto f(b,a), x_3 \mapsto b, x_4 \mapsto a\}$. Por tanto concluimos que el lgg módulo $C$ de $t$*

**Decompose**$_C$

$$\frac{C_f \in ax(f) \wedge A_f \notin ax(f) \wedge i \in \{1,2\}}{\langle f(t_1,t_2) \stackrel{x}{=} f(s_1,s_2) \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle t_1 \stackrel{x_1}{=} s_i \wedge t_2 \stackrel{x_2}{=} s_{(i \bmod 2)+1} \wedge CT \mid S \mid \theta\sigma\rangle}$$

donde $\sigma = \{x \mapsto f(x_1,x_2)\}$, y $x_1, x_2$ son variables frescas

Figura 12.2: Reglas de decomposición para un símbolo de función conmutativo $f$

$$\begin{array}{c}
lgg_E(f(a,b),\ f(b,a)),\ with\ E=\{C_f\} \\
\downarrow\ \text{Configuración Inicial} \\
\langle f(a,b) \stackrel{x}{=} f(b,a) \mid \emptyset \mid id\rangle \\
\swarrow\ \text{Decompose}_C\ \searrow \\
\langle a \stackrel{x_1}{=} b \wedge b \stackrel{x_2}{=} a \mid \emptyset \mid \{x \mapsto f(x_1,x_2)\}\rangle \qquad \langle b \stackrel{x_3}{=} b \wedge a \stackrel{x_4}{=} a \mid \emptyset \mid \{x \mapsto f(x_3,x_4)\}\rangle \\
\downarrow\ \text{Solve} \qquad\qquad\qquad \downarrow\ \text{Decompose} \\
\langle b \stackrel{x_2}{=} a \mid a \stackrel{x_1}{=} b \mid \{x \mapsto f(x_1,x_2)\}\rangle \qquad \langle a \stackrel{x_4}{=} a \mid \emptyset \mid \{x \mapsto f(b,x_4), x_3 \mapsto b\}\rangle \\
\downarrow\ \text{Solve} \qquad\qquad\qquad \downarrow\ \text{Decompose} \\
\langle \emptyset \mid a \stackrel{x_1}{=} b \wedge b \stackrel{x_2}{=} a \mid \{x \mapsto f(x_1,x_2)\}\rangle \qquad \langle \emptyset \mid \emptyset \mid \{x \mapsto f(b,a), x_3 \mapsto b, x_4 \mapsto a\}\rangle \\
\searrow\ minimal_C\ \swarrow \\
\{x \mapsto f(b,a), x_3 \mapsto b, x_4 \mapsto a\}
\end{array}$$

Figura 12.3: Traza del cálculo de la C–generalización de los términos $f(a,b)$ y $f(b,a)$.

*y $s$ es $x\theta = f(b,a)$. Puede observar la traza de ejecución en la Figura 12.3.*

## 12.3. Generalización menos general módulo $A$

En esta sección proporcionamos una regla de inferencia especifica *Decompose*$_A$ para manejar los símbolos de función que obedecen el axioma de asociatividad (pero no el de conmutatividad). Una regla para el manejo de funciones con los axiomas AC se proporciona en la siguiente sección.

La regla *Decompose*$_A$ está dada en la Figura 12.4. Note que utilizamos versiones aplanadas *"flattened"* de los términos que utilizan versiones polivariádicas de los símbolos asociativos, es decir, siendo $f$ un símbolo asociativo,

$$flat(f(t_1,\ldots,f(s_1,\ldots,s_k),\ldots,t_n)) = flat(f(t_1,\ldots,s_1,\ldots,s_k,\ldots,t_n))$$

y, en otro caso, $flat(f(t_1,\ldots,t_n)) = f(flat(t_1),\ldots,flat(t_n))$. Dado un símbolo

**Decompose**$_A$

$$\frac{A_f \in ax(f) \wedge C_f \notin ax(f) \wedge m \geq 2 \wedge n \geq m \wedge k \in \{1, \ldots, (n-m)+1\}}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \stackrel{x}{=} f(s_1, \ldots, s_m) \wedge CT \mid S \mid \theta\rangle \Rightarrow \\ \langle f(t_1, \ldots, t_k) \stackrel{x_1}{=} s_1 \wedge f(t_{k+1}, \ldots, t_n) \stackrel{x_2}{=} f(s_2, \ldots, s_m) \wedge CT \mid S \mid \theta\sigma\rangle\end{array}}$$

donde $\sigma = \{x \mapsto f(x_1, x_2)\}$, y $x_1, x_2$ son variables frescas

Figura 12.4: Regla de decomposición para un símbolo de función $f$ asociativo (no conmutativo).

asociativo $f$ y un término $f(t_1, \ldots, t_n)$ llamosmos *alien $f$-terms* (o simplemente *alien terms*) a aquellos términos entre $t_1, \ldots, t_n$ que no tienen como raíz $f$.

En lo que sigue, siendo $f$ un símbolo asociativo polivariádico, $f(t)$ representa el mismo término $t$, puesto que el símoblo $f$ necesita de al menos dos argumentos. Esta regla de inferencia remplaza a la regla sintáctica de decomposición *Decompose* para el caso de un símbolo de función asociativo $f$, donde se consideran todos los *prefijos* de $t_1, \ldots, t_n$ y $s_1, \ldots, s_m$.

Nótese que esta regla es no determinista, por lo que, todas las posibilidades deben ser exploradas.

Note que esto es mejor que generar todos los términos en la clase de equivalencia correspondiente, como explicamos al principio del Capítulo 12, puesto que pararemos en una constraint de forma perezosa en $t \stackrel{x}{=} f(t_1, \ldots, t_n)$ si $root(t) \not\equiv f$ sin considerar todas las combinaciones de $f(t_1, \ldots, t_n)$.

Damos la regla *Decompose*$_A$ para el caso cuando, en el problema de generalización $s \stackrel{x}{\triangleq} t$, el número de *alien terms* en $s$ es mayor que (o igual a) el número de *alien terms* en $t$. Para el caso de que el número de *alien terms* en $s$ es menor que (o igual a) el número de *alien terms* en $t$, se necesita una regla similar, pero la omitimos puesto que es perfectamente análoga.

**Ejemplo 12.3.1** *Sea $f(f(a, c), b)$ y $f(c, b)$ dos términos donde $f$ es asociativo, es decir, $A_f \in ax(f)$. Aplicando las reglas Solve$_E$, Recover$_E$, y Decompose$_A$ anteriores, terminamos en una configuración final $\langle \emptyset \mid S \mid \theta\rangle$, donde $\theta = \{x \mapsto f(x_3, b), x_4 \mapsto b\}$, por eso concluimos que el lgg módulo $A$ de $t$ y $s$ es $f(x_3, b)$. La traza del cálculo se muestra en la Figura 12.5.*

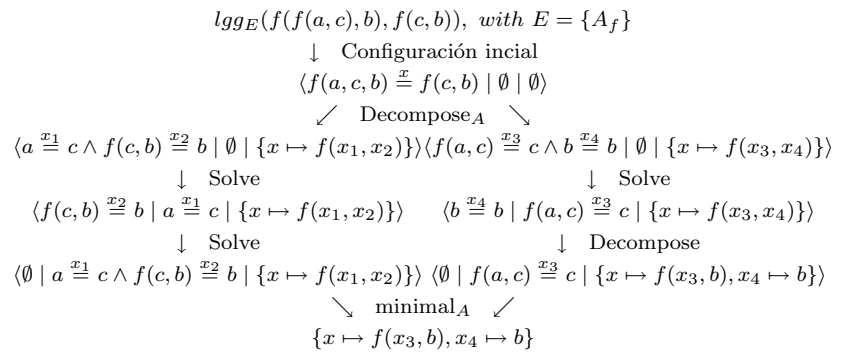


Figura 12.5: Traza del cálculo de la A–generalización de los términos $f(f(a,c),b)$ y $f(c,b))$.

## 12.4. Generalización menos general módulo $AC$

En esta sección proporcionamos una regla escpecífica $Decompose_{AC}$ para el manejo de símbolos de función que obedecen los axiomas de asociatividad y conmutatividad. Note que utilizamos otra vez versiones *flattened* de los términos, como en el caso asociativo de la Sección 12.3.

En realidad, la nueva regla de decomposición para el caso AC es similar a la regla de decomposición para los símbolos de función asociativos, salvo que se consideran todas las permutaciones de $f(t_1,\ldots,t_n)$ y $f(s_1,\ldots,s_m)$. Nótese que esta regla es no determinista, por lo tanto, se deben explorar todas las posibilidades.

De manera similar a la regla $Decompose_A$, damos la regla $Decompose_{AC}$ para el caso cuando, en el problema de generalización $s \stackrel{x}{\triangleq} t$, el número de *alien terms* en $s$ es mayor o igual al número de *alien terms* en $t$. Omitimos la regla para el caso contrario puesto que es perfectamente análoga. Para simplificar, escribimos $\{i_1,\ldots,i_k\}\bar{\uplus}\{i_{k+1},\ldots,i_n\} = \{1,\ldots,n\}$ para denotar que para cada $k,j \in \{1,\ldots,n\}$, $i_j \in \{1,\ldots,n\}$, $\{i_{k+1},\ldots,i_n\} = \{1,\ldots,n\} \setminus \{i_1,\ldots,i_k\}$, $i_1 \leq \cdots \leq i_k$, y $i_{k+1} \leq \cdots \leq i_n$. Note que no es necesario que $i_k \leq i_{k+1}$.

**Ejemplo 12.4.1** *Sea $t = f(a,f(a,b))$ y $s = f(f(b,b),a)$ dos términos donde $f$ es asociativo y conmutativo, es decir, $\{A_f, C_f\} \subseteq ax(f)$. Aplicando las reglas $Solve_E$, $Recover_E$, y $Decompose_{AC}$ anteriores, terminamos en dos configuraciones finales cuyos componentes de las sustituciones son $\theta_1 = \{x \mapsto f(x_1,x_1,x_3), x_2 \mapsto x_1\}$ y $\theta_2 = \{x \mapsto f(x_4,a,b), x_5 \mapsto a, x_6 \mapsto b\}$, por eso decimos que los lggs módulo*

**Decompose**$_{AC}$

$$\frac{\{A_f, C_f\} \subseteq ax(f) \wedge n \geq m \ \wedge \{i_1, \ldots, i_{m-1}\} \bar{\uplus} \{i_m, \ldots, i_n\} = \{1, \ldots, n\}}{\begin{array}{c}\langle f(t_1, \ldots, t_n) \stackrel{x}{=} f(s_1, \ldots, s_m) \wedge C \mid S \mid \theta\rangle \Rightarrow \\ \langle t_{i_1} \stackrel{x_1}{=} s_1 \wedge \ldots \wedge t_{i_{m-1}} \stackrel{x_{m-1}}{=} s_{m-1} \wedge f(t_{i_m}, \ldots, t_{i_n}) \stackrel{x_m}{=} s_m \wedge C \mid S \mid \theta\sigma\rangle\end{array}}$$

donde $\sigma = \{x \mapsto f(x_1, \ldots, x_m)\}$, y $x_1, \ldots, x_m$ son variables frescas

Figura 12.6: Regla de decomposición para un símbolo de función $f$ asociativo-conmutativo.

$$lgg_E(f(a, f(a, b)), f(f(b, b), a)),\ with\ E = \{C_f, A_f\}$$
$$\downarrow \text{ Configuración Inicial}$$
$$\langle f(a, a, b) \stackrel{x}{=} f(b, b, a) \mid \emptyset \mid id\rangle$$
$$\swarrow \text{ Decompose}_{AC} \text{ (No se muestran otras permutaciones) } \searrow$$
$$\langle a \stackrel{x_1}{=} b \wedge a \stackrel{x_2}{=} b \wedge b \stackrel{x_3}{=} a \mid \emptyset \mid \{x \mapsto f(x_1, x_2, x_3)\}\rangle \qquad \langle a \stackrel{x_4}{=} b \wedge a \stackrel{x_5}{=} a \wedge b \stackrel{x_6}{=} b \mid \emptyset \mid \{x \mapsto f(x_4, x_5, x_6)\}\rangle$$
$$\downarrow \text{ Solve} \qquad\qquad \downarrow \text{ Solve}$$
$$\langle a \stackrel{x_2}{=} b \wedge b \stackrel{x_3}{=} a \mid a \stackrel{x_1}{=} b \mid \{x \mapsto f(x_1, x_2, x_3)\}\rangle \qquad \langle a \stackrel{x_5}{=} a \wedge b \stackrel{x_6}{=} b \mid a \stackrel{x_4}{=} b \mid \{x \mapsto f(x_4, x_5, x_6)\}\rangle$$
$$\downarrow \text{ Recover} \qquad\qquad \downarrow \text{ Decompose}$$
$$\langle b \stackrel{x_3}{=} a \mid a \stackrel{x_1}{=} b \mid \{x \mapsto f(x_1, x_1, x_3), x_2 \mapsto x_1\}\rangle \qquad \langle b \stackrel{x_6}{=} b \mid a \stackrel{x_4}{=} b \mid \{x \mapsto f(x_4, a, x_6), x_5 \mapsto a\}\rangle$$
$$\downarrow \text{ Solve} \qquad\qquad \downarrow \text{ Decompose}$$
$$\langle \emptyset \mid a \stackrel{x_1}{=} b \wedge b \stackrel{x_3}{=} a \mid \{x \mapsto f(x_1, x_1, x_3), x_2 \mapsto x_1\}\rangle \qquad \langle \emptyset \mid a \stackrel{x_4}{=} b \mid \{x \mapsto f(x_4, a, b), x_5 \mapsto a, x_6 \mapsto b\}\rangle$$
$$\searrow\ minimal_{AC}\ \swarrow$$
$$\{x \mapsto f(x_1, x_1, x_3), x_2 \mapsto x_1\} \text{ and } \{x \mapsto f(x_4, a, b), x_5 \mapsto a, x_6 \mapsto b\}$$

Figura 12.7: Traza del cálculo de las AC–generalizaciones de los términos $f(a, f(a, b))$ y $f(f(b, b), a)$.

*AC de $t$ y $s$ son $f(x_1, x_1, x_3)$ y $f(x_4, a, b)$. La correspondiente traza del cálculo se muestra en la Figura 12.7.*

## 12.5. Generalización menos general módulo $U$

Finalmente, introducimos la regla de inferencia *Expand*$_U$ de la Figura 12.8 para el manejo de símbolos de función $f$ que tienen un elemento de identidad $e$.

Esta regla considera los axiomas de identidad de una forma perezosa o bajo demanda. La regla corresponde al caso cuando el símbolo de raíz $f$ del término $t$ en la parte izquierda de la constraint $t \stackrel{x}{=} s$ posee el axioma de unidad. Omitimos la regla para el manejo del caso cuando el símbolo raíz del término $s$ en la parte derecha obedece el axioma de unidad, ya que es perfectamente análoga.

**Expand**$_U$

$$\frac{U_f \in ax(f) \wedge root(t) \equiv f \wedge root(s) \not\equiv f \wedge s' \in \{f(e,s), f(s,e)\}}{\langle t \stackrel{x}{=} s \wedge CT \mid S \mid \theta\rangle \Rightarrow \langle t \stackrel{x}{=} s' \wedge CT \mid S \mid \theta\rangle}$$

Figura 12.8: Regla de inferencia para expandir el símbolo de función $f$ con el elemento identidad $e$.

**Ejemplo 12.5.1** *Sea* $t = f(a,b,c,d)$ *y* $s = f(a,c)$ *dos términos donde* $\{A_f, C_f, U_f\} \subseteq ax(f)$. *Aplicando las reglas* $Solve_E$, $Recover_E$, $Decompose_{AC}$, *y* $Expand_U$ *anteriores, terminamos en una configuración final* $\langle \emptyset \mid S \mid \theta\rangle$, *donde* $\theta = \{x \mapsto f(a, f(c, f(x_5, x_6))),\ x_1 \mapsto a, x_2 \mapsto f(c, f(x_5, x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5, x_6)\}$, *por eso tenemos que el lgg módulo ACU de* $t$ *y* $s$ *es* $f(a,c,x_5,x_6)$. *La traza de cálculo se muestra en la Figura 12.9.*

## 12.6. Un método para la ACU-generalización

Para el caso general en el que diferentes símbolos de función satisfacen diferentes axiomas considerados (asociatividad y/o conmutatividad y/o identidad), podemos utilizar todas las anteriores reglas de forma conjunta, sin necesidad de realizar cambios o adaptaciones.

La principal propiedad del conjunto de reglas de inferencia es su *localidad*: se aplican a los símbolos de función dados en el término de la izquierda (o derecha en algunos casos) de la constraint, independientemente de otros símbolos de función puedan estar presentes en la signatura $\Sigma$ y la teoría $E$. Esta localidad significa que estas reglas son *modulares*, en el sentido de que no necesitan ser cambiadas o modificadas cuando nuevos símbolos de función se añaden a la signatura y una nueva función $A$, y/o $C$, y/o $U$ se añade a $E$. Sin embargo, cuando se añaden nuevos axiomas a $E$, algunas reglas que se aplicaban antes (por ejemplo la decomposición para un símbolo $f$ el cual antes satisfacía $ax(f) = \emptyset$, ahora tiene $ax(f) \neq \emptyset$) pueden no aplicarse, y, por el contrario, algunas reglas que no se aplicaban antes ahora pueden aplicarse (porque se añaden nuevos axiomas a $f$). Pero *¡las reglas no cambian!* Son las mismas y pueden ser utilizadas para calcular el conjunto de lggs de dos términos módulo *cualquier* teoría $E$ en la familia *paramétrica* $\mathbb{E}$ de

$$lgg_E(f(a,b,c,d), f(a,c)),\ with\ E = \{C_f, A_f, U_f\}$$
$$\downarrow \quad \text{Configuración Inicial}$$
$$\langle f(a,b,c,d) \stackrel{x}{=} f(a,c) \mid \emptyset \mid id \rangle$$
$$\downarrow \quad \text{Decompose}_{AC} \text{ (No se muestran otras permutaciones)}$$
$$\langle a \stackrel{x_1}{=} a \wedge f(b,c,d) \stackrel{x_2}{=} c \mid \emptyset \mid \{x \mapsto f(x_1,x_2)\} \rangle$$
$$\downarrow \quad \text{Decompose}$$
$$\langle f(b,c,d) \stackrel{x_2}{=} c \mid \emptyset \mid \{x \mapsto f(a,x_2), x_1 \mapsto a\} \rangle$$
$$\downarrow \quad \text{Expand}_U$$
$$\langle f(b,c,d) \stackrel{x_2}{=} f(c,e) \mid \emptyset \mid \{x \mapsto f(a,x_2), x_1 \mapsto a\} \rangle$$
$$\downarrow \quad \text{Decompose}_{AC} \text{ (No se muestran otras permutaciones)}$$
$$\langle c \stackrel{x_3}{=} c \wedge f(b,d) \stackrel{x_4}{=} e \mid \emptyset \mid \{x \mapsto f(a,f(x_3,x_4)), x_1 \mapsto a, x_2 \mapsto f(x_3,x_4)\} \rangle$$
$$\downarrow \quad \text{Decompose}$$
$$\langle f(b,d) \stackrel{x_4}{=} e \mid \emptyset \mid \{x \mapsto f(a,f(c,x_4)), x_1 \mapsto a, x_2 \mapsto f(c,x_4), x_3 \mapsto c\} \rangle$$
$$\downarrow \quad \text{Expand}_U$$
$$\langle f(b,d) \stackrel{x_4}{=} f(e,e) \mid \emptyset \mid \{x \mapsto f(a,f(c,x_4)), x_1 \mapsto a, x_2 \mapsto f(c,x_4), x_3 \mapsto c\} \rangle$$
$$\downarrow \quad \text{Decompose}_{AC} \text{ (No se muestran otras permutaciones)}$$
$$\langle b \stackrel{x_5}{=} e \wedge d \stackrel{x_6}{=} e \mid \emptyset \mid \{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\} \rangle$$
$$\downarrow \quad \text{Solve}$$
$$\langle d \stackrel{x_6}{=} e \mid b \stackrel{x_5}{=} e \mid \{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\} \rangle$$
$$\downarrow \quad \text{Solve}$$
$$\langle \emptyset \mid b \stackrel{x_5}{=} e \wedge d \stackrel{x_6}{=} e \mid \{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\} \rangle$$
$$\downarrow \quad \text{minimal}_{ACU}$$
$$\{x \mapsto f(a,f(c,f(x_5,x_6))), x_1 \mapsto a, x_2 \mapsto f(c,f(x_5,x_6)), x_3 \mapsto c, x_4 \mapsto f(x_5,x_6)\}$$

Figura 12.9: Traza para el cálculo de la U–generalización de los términos $f(a,b,c,d)$ y $f(a,c)$.

las teorías de la forma $E = \bigcup_{f \in \Sigma} ax(f)$, donde $ax(f) \subseteq \{A_f, C_f, U_f\}$.

El algoritmo para el cálculo de la menor generalización menos general ACU presentado, ha sido desarrollado en Maude [Clavel et al., 2007], con una interfaz web escrita en Java. El núcleo de la herramienta contiene alrededor de 200 líneas de código Maude. La herramienta Web que se muestra en la figura 12.10 una captura de pantalla, está públicamente disponible junto con un conjunto de ejemplos en la siguiente url:

`http://www.dsic.upv.es/users/elp/toolsMaude/Lgg.html`.

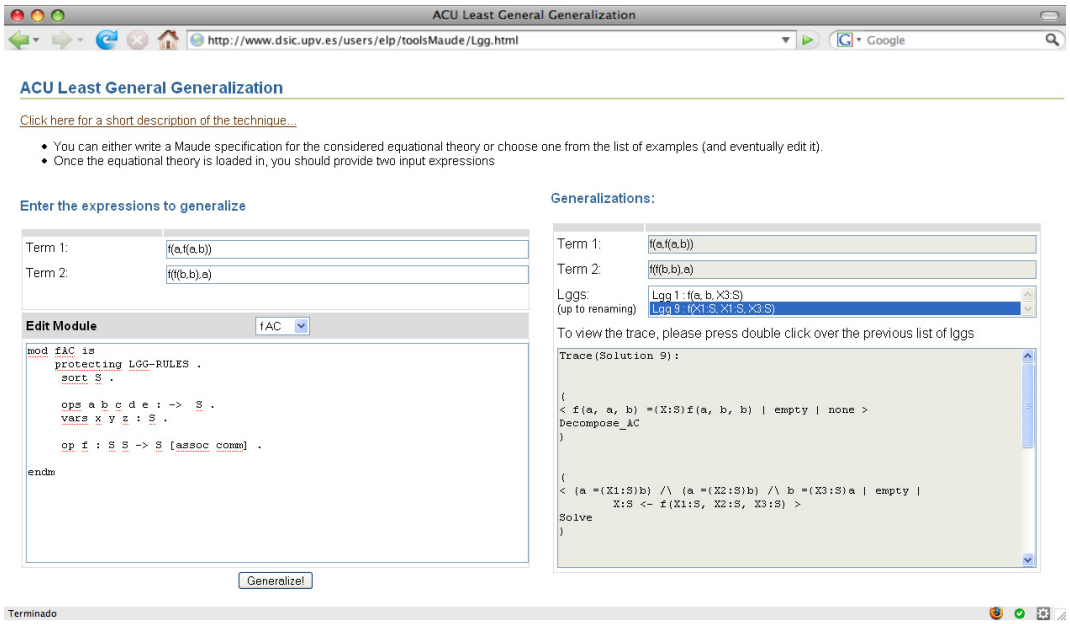


Figura 12.10: Captura de pantalla de la interfaz Web.

# 13

# Generalización Order-Sorted

En este Capítulo, generalizamos al *order-sorted* el algoritmo sin tipos o sintáctico presentado en el Capítulo 11.

En este Capítulo, consideramos que los términos a generalizar $t$ y $t'$ tienen el mismo *top sort*; en otro caso son incomparables y no existe la generalización. Para obtener el lgg, empezamos desde una configuración inicial $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id \rangle$ donde $[\mathsf{s}] = [LS(t)] = [LS(t')]$, y vamos transformando las configuraciones hasta que alacanzamos una configuración final $\langle \emptyset \mid S \mid \theta \rangle$. En el caso *order–sorted* el lgg no es en general único. Cada configuración final $\langle \emptyset \mid S \mid \theta \rangle$ proporciona un lgg de $t$ y $t'$ dado por $(x{:}[\mathsf{s}])\theta$.

La relación de transición $\rightarrow$ viene en este por la menor relación que satisface las reglas de la Figura 13.1. El significado de las reglas es el siguiente:

- La regla **Decompose** es la decomposición sintáctica que genera nuevas constraints a ser resueltas. Las variables frescas son inicialmente asignadas al *top sort*, el cual se "degradará" cuando sea necesario.
- La regla **Recover** es similar a la regla correspondiente a la Figura 11.1.
- La regla **Solve** comprueba si una constraint $t \overset{y}{\triangleq} t' \in C$, con $root(s) \not\equiv root(t)$, no está ya resulta. En dicho caso, la *constraint* resuelta $t \overset{y}{\triangleq} t'$ se añade al *store* $S$, y la sustitución $\{x \mapsto z\}$ se compone con la parte de sustitutción acarreada, donde $z$ es una variable fresca con *sort* en el $LUBS$ del menor *sort* de ambos términos. Note que esté es el único no determinismo (además de la elección de la constraint) en nuestras reglas de inferencia, en contraste con la Figura 11.1. Este no determinismo extra provoca que nuestras reglas no sean confluentes en general.

**Decompose**
$$\frac{f \in (\Sigma \cup \mathcal{X}) \wedge f : [\mathsf{s_1}] \times \ldots \times [\mathsf{s_n}] \rightarrow [\mathsf{s}]}{\langle f(t_1, \ldots, t_n) \overset{x:[\mathsf{s}]}{\triangleq} f(s_1, \ldots, s_n) \wedge C \mid S \mid \theta \rangle \rightarrow \langle t_1 \overset{x_1:[\mathsf{s_1}]}{\triangleq} s_1 \wedge \ldots \wedge t_n \overset{x_n:[\mathsf{s_n}]}{\triangleq} s_n \wedge C \mid S \mid \theta\sigma \rangle}$$

donde $\sigma = \{x{:}[\mathsf{s}] \mapsto f(x_1{:}[\mathsf{s_1}], \ldots, x_n{:}[\mathsf{s_n}])\}$, $x_1{:}[\mathsf{s_1}], \ldots, x_n{:}[\mathsf{s_n}]$ son variables frescas, y $n \geq 0$

**Solve**
$$\frac{root(t) \not\equiv root(t') \wedge s' \in LUBS(LS(t), LS(t')) \wedge \nexists y \nexists \mathsf{s''} : t \overset{y:\mathsf{s''}}{\triangleq} t' \in S}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \mid \theta \rangle \rightarrow \langle C \mid S \wedge t \overset{z:\mathsf{s'}}{\triangleq} t' \mid \theta\sigma \rangle}$$

donde $\sigma = \{x{:}[\mathsf{s}] \mapsto z{:}\mathsf{s'}\}$ y $z{:}\mathsf{s'}$ es una variable fresca.

**Recover**
$$\frac{root(t) \not\equiv root(t')}{\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \wedge C \mid S \wedge t \overset{y:\mathsf{s'}}{\triangleq} t' \mid \theta \rangle \rightarrow \langle C \mid S \wedge t \overset{y:\mathsf{s'}}{\triangleq} t' \mid \theta\sigma \rangle}$$

donde $\sigma = \{x{:}[\mathsf{s}] \mapsto y{:}\mathsf{s'}\}$

Figura 13.1: Reglas para la generalización menos general *order–sorted*.

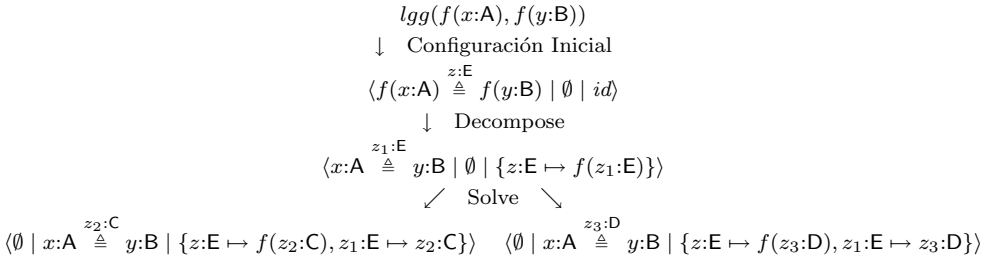


Figura 13.2: Traza del cálculo de la generalización *order–sorted* de los términos $f(x)$ y $f(y)$

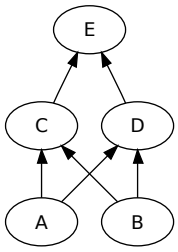


Figura 13.3: Jerarquía de *sorts*

**Ejemplo 13.0.1** *Sea* $t = f(x{:}\mathsf{A})$ *y* $s = f(y{:}\mathsf{B})$ *dos términos donde* $x$ *u* $y$ *son variables de los* sorts $A$ *y* $B$ *respectivamente, y consideremos la jerarquía de* sorts *que se muestra en la Figura 13.3. La definición de tipos de* $f$ *es* $f : \mathsf{E} \to \mathsf{E}$. *Empezando a partir de la configuración inicial* $\langle f(x{:}\mathsf{A}) \stackrel{z:\mathsf{E}}{\triangleq} f(y{:}\mathsf{B}) \mid \emptyset \mid id \rangle$, *aplicamos las reglas de inferencia de la Figura 13.1 y por medio del algoritmo de lgg obtenemos las sustituciones* $\theta_1 = \{z{:}\mathsf{E} \mapsto f(z_2{:}\mathsf{C}), z_1{:}\mathsf{E} \mapsto z_2{:}\mathsf{C}\}$ *y* $\theta_2 = \{z{:}\mathsf{E} \mapsto f(z_3{:}\mathsf{D}), z_1{:}\mathsf{E} \mapsto z_3{:}\mathsf{D}\}$. *Por consiguiente, el lgg es o bien* $(z{:}\mathsf{E})\theta_1 = f(z_2{:}\mathsf{C})$ *o* $(z{:}\mathsf{E})\theta_2 = f(z_3{:}\mathsf{D})$. *Note que* $\theta_1$ *y* $\theta_2$ *son incomparables, por lo que tenemos dos posibles lggs. La traza de ambas soluciones se muestran en la Figura 13.2.*

Antes de probar la corrección del sistema de reglas anterior, damos una caracterización abstracta del conjunto de lggs de dos términos $t$ y $t'$ tal que $[LS(t)] = [LS(t')]$. Para simplificar nuestra notación, en lo que sigue, escribimos $t[s]_{p_1,\ldots,p_n}$ en vez de $((t[s]_{p_1})\cdots)[s]_{p_n}$.

**Definición 13.0.2** *Dados los términos* $t$ *y* $t'$ *tales que* $[LS(t)] = [LS(t')]$, *sean* $(u_1, v_1), \ldots, (u_k, v_k)$ *los pares conflictivos de* $t$ *y* $t'$. *Para cada par conflictivo* $(u_i, v_i)$, $p^i_1, \ldots, p^i_{n_i}$ *para denotar a las posiciones conflictivas, y asumiremos que* $[\mathsf{s_i}] = [LS(u_i)] = [LS(v_i)]$. *Definimos los términos* $lgg^\bullet(t, t') = ((t[x_1{:}[\mathsf{s_1}]]_{p^1_1,\ldots,p^1_{n_1}})\cdots)[x_k{:}[\mathsf{s_k}]]_{p^k_1,\ldots,p^k_{n_k}}$, *donde* $x_1{:}[\mathsf{s_1}], \ldots, x_k{:}[\mathsf{s_k}]$ *son variables frescas. Además, definimos*

$$Spec(t, t') = \{\rho \mid Dom(\rho) = \{x_1{:}[\mathsf{s_1}], \ldots, x_k{:}[\mathsf{s_k}]\} \wedge \forall 1 \leq i \leq k, \rho(x_i{:}[\mathsf{s_i}]) =$$

$$x_i{:}\mathsf{s}'_\mathsf{i} \wedge \mathsf{s}'_\mathsf{i} \in LUBS(LS(u_i), LS(v_i))\}$$

*donde todas las* $x_i{:}\mathsf{s}'_\mathsf{i}$ *son variables frescas, y, finalmente,* $lgg(t, t') = \{lgg^\bullet(t, t')\rho \mid \rho \in Spec(t, t')\}$.

**Lema 13.0.3** *Dados los términos $t$ y $t'$ tal que $[LS(t)] = [LS(t')]$, $lgg^{\bullet}(t,t')$ es una generalización de $t$ y $t'$ y $lgg(t,t')$ proporciona un conjunto completo minimal de lggs.*

Proporcionamos algunas notaciones auxiliares y lemas.

**Lema 13.0.4** *Dados los términos $t$ y $t'$ tales que $[\mathsf{s}] = [LS(t)] = [LS(t')]$, y una variable fresca $x{:}[\mathsf{s}]$ tal que $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$. Entonces, una constraint $u \overset{z}{\triangleq} v$ está en $S$ si y solo si existe un posición conflictiva $p$ de $t$ y $t'$ tal que $t|_p = u$ y $t'|_p = v$, y existe un nombre de variable $y$ de un* sort *$s \in LUBS(LS(u), LS(v))$ tal que $z = y{:}s$.*

Una sustitución $\delta$ se llama *"degradada"* si cada remplazamiento es de la forma $x{:}\mathsf{s} \mapsto x'{:}\mathsf{s}'$, donde $x$ y $x'$ son variables y $\mathsf{s}' \leq \mathsf{s}$.

**Lema 13.0.5** *Consideremos los términos $t$ y $t'$ tales que $[\mathsf{s}] = [LS(t)] = [LS(t')]$, y sea $lgg^{\bullet}(t,t')$. Entonces, para todo $S$ y $\theta$ tal que $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$, existe una sustitución "degradada" $\delta$ tal que $lgg^{\bullet}(t,t')\delta = (x{:}[\mathsf{s}])\theta$.*

**Teorema 13.0.6** *Dados los términos $t$ y $t'$ tal que $[\mathsf{s}] = [LS(t)] = [LS(t')]$, y una variable fresca $x{:}[\mathsf{s}]$, el término $u \in lgg(t,t')$ es un lgg de $t$ y $t'$ si y solo si $\langle t \overset{x:[\mathsf{s}]}{\triangleq} t' \mid \emptyset \mid id\rangle \rightarrow^* \langle \emptyset \mid S \mid \theta\rangle$ para algún $S$ y $\theta$ y hay un renombramiento $\rho$ tal que $u\rho = (x{:}[\mathsf{s}])\theta$.*

**Proof.** Razonamos por contradicción. Ambos casos del "*si* y *solo si*" son similares. Proporcionamos solo la prueba par el caso *si*.
Asumamos algún $S$ y $\theta$ tal que no hay $u \in lgg(t,t')$ y un renombramiento $\rho$ tal que $u\rho = (x{:}[\mathsf{s}])\theta$. Para todo $u \in lgg(t,t')$, por la Definición 13.0.2, $lgg^{\bullet}(t,t') \leq u$ con una sustitución "degradada". Por el Lema 13.0.5, $lgg^{\bullet}(t,t') \leq (x{:}[\mathsf{s}])\theta$ con una sustitución "degradada". Sea $\delta$ la sustitución "degradada" tal que $lgg^{\bullet}(t,t')\delta = (x{:}[\mathsf{s}])\theta$. Para todo $u \in lgg(t,t')$, sea $\delta_u$ una sustitución "degradada" tal que $lgg^{\bullet}(t,t')\delta_u = u$. Puesto que no hay renombramiento entre $(x{:}[\mathsf{s}])\theta$ y $u$ y ambos tienen una sustitución "degradada" con $lgg^{\bullet}(t,t')$, debe existir un remplazamiento $x{:}\mathsf{s} \mapsto x'{:}\mathsf{s}'$ en $\delta$ y un remplazamiento $x{:}\mathsf{s} \mapsto x''{:}\mathsf{s}''$ en $\delta_u$ tal que o bien $\mathsf{s}' < \mathsf{s}''$, $\mathsf{s}'' < \mathsf{s}'$, o $[\mathsf{s}'] \neq [\mathsf{s}'']$. Pero las tres posibilidades son imposibles por definición, puesto que $\mathsf{s}' < \mathsf{s}''$ contradice la idea que $u$ es un lgg, $\mathsf{s}'' < \mathsf{s}'$ contradice el Lema 13.0.4, y $[\mathsf{s}'] \neq [\mathsf{s}'']$ contradice ambos, que $u$ es un lgg y el Lema 13.0.4. □

# 14

# Conclusiones y Trabajos futuros

En esta segunda parte de la tesis, hemos redefinido el algoritmo de Huet, mediante un sistema de reglas de inferencia, el cual es la base de los algoritmos de generalización ecuacional y *order-sorted* cuya corrección ha sido también probada.

Se ha presentado un nuevo algoritmo de generalización ecuacional que calcula un conjunto minimal y completo de generalizaciones menos generales para dos términos módulo cualquier combinación de los axiomas de asociatividad, conmutatividad e identidad para símbolos binarios en una teoría. Además hemos proporcionado una herramienta que implementa este algoritmo de generalización ecuacional y que está disponible públicamente.

Por otra parte, hemos presentado un algoritmo de generalización order-sorted que calcula un conjunto completo y minimal de lggs para dos términos en presencia de *sorts*.

Nuestros algoritmos son aplicables directamente a cualquier lenguaje declarativo *many-sorted*, *order-sorted* (estos dos para el caso del algoritmo *order-sorted*) y a sistemas de razonamiento. También son aplicables a lenguajes sin tipos y sistemas que tengan sólo un *sort*. Sin embargo, varios de estos lenguajes – cómo ASF+SDF, OBJ, Cafe-OBJ, Elan, y Maude –, así como diversos demostradores de teoremas, también soportan frecuentemente la ocurrencia de axiomas ecuacionales tales como la asociatividad, conmutatividad e identidad.

## Trabajo Futuro

En un primer trabajo, queremos realizar un estudio de complejidad algorítmica y funcional de nuestros algoritmos de generalización.

Sería muy útil la combinación de los sistemas de inferencia *order-sorted* y

$E$–generalización en un sólo cálculo que soportará ambas generalizaciones. Sin embargo, esta combinación nos parece no trivial y queda como trabajo futuro, ya que que queremos utilizar el algoritmo que resulte de la unión de los algoritmos ecuacional y *order-sorted* como una parte importante de un evaluador parcial basado en narrowing, para programas en lenguajes basados en reglas tales como OBJ, CafeOBJ, y Maude.

# Bibliografía

[Aït-Kaci and Sasaki, 2001] Aït-Kaci, H. and Sasaki, Y.: 2001, An axiomatic approach to feature term generalization, in *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*, pp 1–12, Springer-Verlag, London, UK

[Alpuente et al., 2007a] Alpuente, M., Ballis, D., Escobar, S., Falaschi, M., Ojeda, P., and Romero, D.: 2007a, *Un motor algebraico para la verificación de sistemas Web en Gverdi*, Technical Report DSIC-II/02/07, DSIC, UPV

[Alpuente et al., 2006a] Alpuente, M., Ballis, D., and Falaschi, M.: 2006a, *Software Tools for Technology Transfer* **8(6)**, 565

[Alpuente et al., 2007b] Alpuente, M., Ballis, D., Falaschi, M., Ojeda, P., and Romero, D.: 2007b, A Fast Algebraic Web Verification Service, in *Proc. of First Int'l Conf. on Web Reasoning and Rule Systems (RR 2007)*, Vol. 4524 of *Lecture Notes in Computer Science*, pp 239–248

[Alpuente et al., 2007c] Alpuente, M., Ballis, D., Falaschi, M., Ojeda, P., and Romero, D.: 2007c, Estrategias de Reparación para Sitios Web Incompletos, in *XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007)*

[Alpuente et al., 2007d] Alpuente, M., Ballis, D., Falaschi, M., Ojeda, P., and Romero, D.: 2007d, The Web Verification Service WebVerdi-M, in *VII Jornadas sobre PROgramación y LEnguajes (PROLE 2007)*, pp 93–102

[Alpuente et al., 2007e] Alpuente, M., Ballis, D., Falaschi, M., Ojeda, P., and Romero, D.: 2007e, *The Web Verification Service WebVerdi-M*, Technical Report DSIC-II/08/07, DSIC-UPV

[Alpuente et al., 2008] Alpuente, M., Ballis, D., Falaschi, M., Ojeda, P., and Romero, D.: 2008, An Abstract Generic Framework for Web Sites Verifica-

tion, in *2008 International Symposium on Applications and the Internet*, IEEE Computer Society

[Alpuente et al., 2006b] Alpuente, M., Ballis, D., Falaschi, M., and Romero, D.: 2006b, A Semi-automatic Methodology for Reparing Faulty Web Sites, in *Proc. of the 4th IEEE Int'l Conference on Software Engineering and Formal Methods (SEFM'06)*, pp 31–40, IEEE Computer Society Press

[Baader and Nipkow, 1998] Baader, F. and Nipkow, T.: 1998, *Term Rewriting and All That*, Cambridge University Press

[Baader and Snyder, 1999] Baader, F. and Snyder, W.: 1999, Unification theory, in *Handbook of Automated Reasoning*, Elsevier

[Ballis, 2005] Ballis, D.: 2005, *Ph.D. thesis*, University of Udine and Technical University of Valencia

[Ballis and Vivó, 2005] Ballis, D. and Vivó, J. G.: 2005, A Rule-based System for Web Site Verification, in *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, Vol. 157(2), ENTCS, Elsevier

[Bergstra et al., 1989] Bergstra, J., Heering, J., and Klint, P.: 1989, *Algebraic Specification*, ACM Press

[Bertossi and Pinto, 1999] Bertossi, L. and Pinto, J.: 1999, Specifying Active Rules for Database Maintenance, in G. Saake, K. Schwarz, and C. Türker (eds.), *Transactions and Database Dynamics, 8th Int'l Workshop on Foundations of Models and Languages for Data and Objects*, Vol. 1773 of *Lecture Notes in Computer Science*, pp 112–129, Springer

[Bezem, 2003] Bezem, M.: 2003, *TeReSe, Term Rewriting Systems*, Chapt. Mathematical background (Appendix A), Cambridge University Press

[Borovanský et al., 2002] Borovanský, P., Kirchner, C., Kirchner, H., and Moreau, P.-E.: 2002, *Theoretical Computer Science* **285**, 155

[Boyer and Moore, 1980a] Boyer, R. and Moore, J.: 1980a, *A Computational Logic*, Academic Press

[Boyer and Moore, 1980b] Boyer, R. and Moore, J.: 1980b, *A Computational Logic*, in [Boyer and Moore, 1980a]

[Brogi et al., 2004] Brogi, A., Canal, C., Pimentel, E., and Vallecillo, A.: 2004, *Electr. Notes Theor. Comput. Sci.* **105**, 73

[Buneman et al., 2003] Buneman, P., Grohe, M., and Koch, C.: 2003, Path Queries on Compressed XML, in *Proceedings of the 29th Int'l Conference on Very Large Data Bases (VLDB'03)*, pp 141–152, Morgan Kaufmann

[Capra et al., 2002] Capra, L., Emmerich, W., Finkelstein, A., and Nentwich, C.: 2002, *ACM Transactions on Internet Technology* **2(2)**, 151

[Centrum voor Wiskunde en Informatica, 2001] Centrum voor Wiskunde en Informatica: 2001, *XMark – an XML Benchmark Project*, Available at: `http://monetdb.cwi.nl/xml/`

[Clavel et al., 2007] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C.: 2007, *All About Maude - A High-Performance Logical Framework*, Springer-Verlag New York, Inc., Secaucus, NJ, USA

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C.: 2001, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, USA, 2nd edition

[Cortier et al., 2006] Cortier, V., Delaune, S., and Lafourcade, P.: 2006, *Journal of Computer Security* **14(1)**, 1

[Cousot and Cousot, 1977] Cousot, P. and Cousot, R.: 1977, Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints., in *POPL*, pp 238–252

[Dershowitz and Plaisted., 2001] Dershowitz, N. and Plaisted., D.: 2001, *Handbook of Automated Reasoning* **1**, 535

[Diaconescu and Futatsugi, 1998] Diaconescu, R. and Futatsugi, K.: 1998, *CafeOBJ Report*, Vol. 6 of *AMAST Series in Computing*, World Scientific, AMAST Series

[Eker et al., 2003] Eker, S., Meseguer, J., and Sridharanarayanan, A.: 2003, The Maude LTL model checker and its implementation, in *Model Checking Software: Proc. 10 th Intl. SPIN Workshop*, Vol. 2648 of *LNCS*, pp 230–234, Springer

[Escobar et al., 2006a] Escobar, S., Meadows, C., and Meseguer, J.: 2006a, *Theoretical Computer Science* **367(1-2)**, 162

[Escobar et al., 2006b] Escobar, S., Meadows, C., and Meseguer, J.: 2006b, *Theoretical Computer Science* **367(1-2)**, 162

[Escobar et al., 2007] Escobar, S., Meseguer, J., and Thati, P.: 2007, *Electr. Notes Theor. Comput. Sci.* **177**, 5

[Ferrari et al., 2004] Ferrari, G., Gnesi, S., Montanari, U., Raggi, R., Trentanni, G., and Tuosto, E.: 2004, Verication on the Web of mobile systems, in *2nd International Workshop on Verification and Validation of Enterprise Information Systems (VVEIS 2004)*

[Gallagher, 1993] Gallagher, J. P.: 1993, Tutorial on specialisation of logic programs, in *PEPM '93: Proceedings of the 1993 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pp 88–98, ACM, New York, NY, USA

[Goguen et al., 2000] Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., and Jouannaud, J.-P.: 2000, Introducing OBJ, in *Software Engineering with OBJ: Algebraic Specification in Action*, pp 3–167, Kluwer

[Haydar et al., 2004] Haydar, M., Patrenko, A., and Sahraoui, H.: 2004, Formal verification of web applications modeled by communicating automata, in *4th IFIP WG 6.1 Int'l Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004)*

[Huet, 1976] Huet, G.: 1976, *Ph.D. thesis*, Univ. Paris VII

[Imagiware, ] Imagiware, I., *Doctor HTML: Quality Assessment for the Web*, Available at: `http://www.doctor-html.com/RxHTML/`

[Kadhi and El-Gendy, 2006] Kadhi, N. E. and El-Gendy, H.: 2006, *Journal of Computational Methods in Science and Engineering* **6**, 109

[Kaufmann et al., 2000a] Kaufmann, M., Manolios, P., and Moore, J.: 2000a, *Computer-Aided Reasoning: An Approach*, in [Kaufmann et al., 2000b]

[Kaufmann et al., 2000b] Kaufmann, M., Manolios, P., and Moore, J.: 2000b, *Computer-Aided Reasoning: An Approach*, Kluwer

[Kirchner et al., 2005] Kirchner, C., Kirchner, H., and Anderson, S.: 2005, Anchoring modularity in html, in *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, Vol. 157(2), ENTCS, Elsevier

[Klop, 1992] Klop, J.: 1992, Term Rewriting Systems, in S. Abramsky, D. Gabbay, and T. Maibaum (eds.), *Handbook of Logic in Computer Science*, Vol. I, pp 1–112, Oxford University Press

[Lassez et al., 1988] Lassez, J.-L., Maher, M. J., and Marriott, K.: 1988, Unification Revisited, in J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, pp 587–625, Morgan Kaufmann, Los Altos, Ca.

[Legall et al., 2006] Legall, T., Jeannet, B., and Jhon, T.: 2006, Verification of Communication Protocols Using Abstract Interpretation of FIFO Queues, in *Proceedings of Algebraic Methodology and Software Technology, 11th International Conference (AMAST'06)*, Vol. 4019 of *Lecture Notes in Computer Science*, pp 263–274, Springer

[Leuschel, 2002] Leuschel, M.: 2002, Homeomorphic Embedding for Online Termination of Symbolic Methods, in T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough (eds.), *The Essence of Computation*, Vol. 2566 of *LNCS*, pp 379–403, Springer

[Lucas, 2005] Lucas, S.: 2005, Rewriting-Based Navigation of Web Sites: Looking for Models and Logics, in *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, Vol. 157(2), ENTCS, Elsevier

[Martí-Oliet and Meseguer, 2002] Martí-Oliet, N. and Meseguer, J.: 2002, *Theoretical Computer Science* **285(2)**, 121

[Mayol and Teniente, 1999] Mayol, E. and Teniente, E.: 1999, A Survey of Current Methods for Integrity Constraint Maintenance and View Updating, in *Proc. of Advances in Conceptual Modeling: ER '99*, Vol. 1727 of *Lecture Notes in Computer Science*, pp 62–73, Springer

[Meseguer, 1992] Meseguer, J.: 1992, *Theor. Comput. Sci.* **96(1)**, 73

[Meseguer, 1997] Meseguer, J.: 1997, Membership algebra as a logical framework for equational specification, in *WADT*, pp 18–61

[Mogensen, 2000] Mogensen, T. Æ.: 2000, *Higher-Order and Symbolic Computation* 13(4)

[Muggleton, 1999] Muggleton, S.: 1999, *Artif. Intell.* **114(1-2)**, 283

[Nentwich et al., 2003] Nentwich, C., Emmerich, W., and Finkelstein, A.: 2003, Consistency Management with Repair Actions, in *Proc. of the 25th International Conference on Software Engineering (ICSE'03)*, IEEE Computer Society

[Nesbit, 2000] Nesbit, S.: 2000, *HTML Tidy: Keeping it clean*, Available at `http://www.webreview.com/2000/06-16/webauthors/06-16-00-3.shtml`

[Østvold, 2004] Østvold, B.: 2004, *A functional reconstruction of anti-unification*, Technical Report DART/04/04, Norwegian Computing Center, Available at http://publications.nr.no/nr-notat-dart-04-04.pdf

[Pfenning, 1991] Pfenning, F.: 1991, Unification and anti-unification in the calculus of constructions, in *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science, 15-18 July, 1991, Amsterdam, The Netherlands*, pp 74–85, IEEE Computer Society

[Plotkin, 1970] Plotkin, G.: 1970, A note on inductive generalization, in *Machine Intelligence*, Vol. 5, pp 153–163, Edinburgh University Press

[Plotkin, 2004] Plotkin, G.: 2004, *J. Log. Algebr. Program.* **60-61**, 17

[Polyzotis et al., 2004] Polyzotis, N., Garofalakis, M., and Ioannidis, Y. E.: 2004, Approximate XML Query Answers, in *Proceedings of the ACM SIGMOD International Conference on Management of Data (ICMD'04)*, pp 263–274, ACM

[Popplestone, 1969] Popplestone, R.: 1969, An experiment in automatic induction, in *Machine Intelligence*, Vol. 5, pp 203–215, Edinburgh University Press

[Reynolds, 1970] Reynolds, J.: 1970, *Machine Intelligence* **5**, 135

[Scheffczyk et al., 2004a] Scheffczyk, J., , Rödig, U. M. B. P., and Schmitz, L.: 2004a, S-dags: Towards efficient document repair generation, in *Proc. 2nd Int. Conf. on Computing, Communications and Control Technologies*, Vol. 2, pp 308–313

[Scheffczyk et al., 2003] Scheffczyk, J., Borghoff, U. M., Rödig, P., and Schmitz, L.: 2003, Consistent document engineering: formalizing type-safe consistency rules for heterogeneous repositories, in *Proc. of the 2003 ACM Symposium on Document Engineering (DocEng '03)*, pp 140–149, ACM Press

[Scheffczyk et al., 2004b] Scheffczyk, J., Rödig, P., Borghoff, U. M., and Schmitz, L.: 2004b, Managing inconsistent repositories via prioritized repairs, in *Proc. of the 2004 ACM Symposium on Document Engineering (DocEng '04)*, pp 137–146, ACM Press

[Shiriram, 2005] Shiriram, K.: 2005, Web verification: Perspectives and challenges, in *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, Vol. 157(2), ENTCS, Elsevier

[Siekmann, 1989] Siekmann, J.: 1989, *Journal of Symbolic Computation* **7**, 207

[SourceForge, 2005] SourceForge: 2005, *Tri Active JDO*, Available at: `http://tjdo.sourceforge.net`

[Stone, 2005] Stone, R. G.: 2005, Validating scripted web-pages, in *Proc. of 1st Int'l Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, Vol. 157(2), ENTCS, Elsevier

[Sun Microsystems, ] Sun Microsystems, *JavaTM 2 SDK, Standard Edition Documentation, Version 1.4.2*, Available at: `http://java.sun.com`

[Sun Microsystems, 2006] Sun Microsystems: 2006, *JSR 12: JavaTM Data Objects (JDO) Specification*, Available at: `http://www.jcp.org/en/jsr/detail?id=12`

[TeReSe, 2003] TeReSe (ed.): 2003, *Term Rewriting Systems*, Cambridge University Press, Cambridge

[World Wide Web Consortium, 1999] World Wide Web Consortium: 1999, *Extensible Markup Language (XML) 1.0, Second Edition*, Available at: `http://www.w3.org`

[World Wide Web Consortium, 2000] World Wide Web Consortium: 2000, *Extensible HyperText Markup Language (XHTML)*, Available at: `http://www.w3.org`

[World Wide Web Consortium, 2003] World Wide Web Consortium: 2003, *SOAP Version 1.2*, Available at: `http://www.w3.org`

# A

# Lista de publicaciones

- **Internacionales**

  - M. Alpuente, S. Escobar, J. Meseguer, P. Ojeda
    Order-sorted Generalization.
    17th Int'l Workshop on Functional and (Constraint) Logic Programming, WFLP 2008. July 3-4, 2008 – Siena, Italy.
    Electronic Notes in Theoretical Computer Science, To appear 2008. Elsevier

  - M. Alpuente, S. Escobar, J. Meseguer, P. Ojeda
    A Modular Equational Generalization Algorithm
    18th Int'l Symp. on Logic Based Program Synthesis and Tranformation LOPSTR 2008, Valencia (Spain), Jul 2008

  - M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda y D. Romero
    A Fast Algebraic Web Verification Service.
    Proc. of First Int'l Conf. on Web Reasoning and Rule Systems (RR 2007), Innsbruck, Austria.
    *Lecture Note in Computer Science* 4524: 239–248, 2007.

  - M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda y D. Romero
    An Abstract Generic Framework for Web Sites Verification.
    The IEEE/IPSJ Symposium on Applications and the Internet, (SAINT 2008).
    Julio 28, 2008 – Turku, Finland.
    IEEE Computer Society.

- **Nacionales**

- M. Alba, M. Alpuente, S. Escobar, P. Ojeda, D. Romero
  A Tool for Automated Certification of Java Source Code in Maude.
  VIII Jornadas sobre PROgramación y LEnguajes (PROLE 2008), en el marco del Congreso Español de Informática CEDI'08, Gijón, Octubre 2008, *to appear*.
- M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero
  Estrategias de Reparación para Sitios Web Incompletos.
  XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007).
  Octubre 1–5, 2007 – Corrientes, Argentina.
- M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero
  The Web Verification Service WebVerdi-M.
  VII Jornadas sobre PROgramación y LEnguajes (PROLE 2007), en el marco del Congreso Español de Informática CEDI'07, Zaragoza, Septiembre 2007, pp 93–102.

- **Informes Técnicos (No publicados)**

  - M. Alpuente, D. Ballis, S. Escobar, M. Falaschi, P. Ojeda y D. Romero
    Un motor algebraico para la verificación de sistemas Web en Gverdi
    Technical Report DSIC-II/02/07. DSIC, UPV. Enero 18, 2007.
  - M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda y D. Romero
    The Web Verification Service WebVerdi-M
    Technical Report DSIC-II/08/07. DSIC, UPV. Marzo, 2007.
  - M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda y D. Romero
    Abstract Web Site Verification in WebVerdi-M
    Technical Report DSIC-II/19/07. DSIC, UPV. Noviembre, 2007.
  - M. Alpuente, S. Escobar, J. Mesesguer y P. Ojeda
    Order-Sorted Generalization
    Technical Report DSIC-II/05/08. DSIC, UPV. Mayo, 2008.
  - M. Alpuente, S. Escobar, J. Mesesguer y P. Ojeda
    A Modular Equational Generalization Algorithm
    Technical Report DSIC-II/06/08. DSIC, UPV. Mayo, 2008.

- **Otras contribuciones**

  - Red Maude.
    A Modular Equational Generalization Algorithm.
    Valencia, Julio 14, 2008.
  - ICT for EU-India Cross Cultural Dissemination.
    Web Verdi-M (An advisory system for web verification using Maude).
    Udine - Italia, Diciembre 13, 2006.