Universidad Politécnica de Valencia

Departamento de Sistemas
Informáticos y Computación

Tesis de Master

Métodos Formales en la Verificación y Reparación de Sitios Web

Estrategias, Implementación y Análisis estático

Candidato:
Daniel Romero

Director:
María Alpuente Frasnedo

– Julio de 2007 –

Trabajo realizado con una beca del proyecto
ALFA LERNet AML/19.0902/97/0666/II-0472-FA

Correo Electrónico del autor: dromero@dsic.upv.es

Dirección del autor:

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

... el zar Iván IV, apodado el Terrible,
propuso cierta vez un problema
a un geómetra de su corte.

Consistía en determinar
cuántos ladrillos serían necesarios
para la construcción de un edificio regular
cuyas dimensiones se indicaban.
La respuesta fue rápida y
la construcción realizada demostró más tarde
la exactitud de los cálculos.

Iván, impresionado con este hecho,
mandó a quemar al matemático
convencido de que haciendo esto,
libraba al pueblo ruso de
un peligroso hechicero.

Resumen

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 de verificación y reparación de sitios Web. Esta metodología 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 de esta tesis son:

- Presentamos una metodología de reparación simi-automática de sitios Web que permite obtener un sitio Web correcto y completo con respecto a su espcificación.

- 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 de [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 accesible las capacidades de verificación Web propuestas.
- Exploramos una aplicación de la técnica de reescritura parcial para el filtrado de documentos XML.

Índice general

Índice de figuras

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 hoy en día no es cuestionado 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 nos interesa 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., 2007f; 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 in-

tegridad 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., 2007f]. 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 mas servicios Web son compatibles para interoperar o no. En caso de no serlo, generará de manera automática, cuando es posible, la especificación del adaptador que medie 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 esta 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; Ba-

llis and Vivó, 2005], desarrollado inicialmente en Haskell. Una vez realizada la verificación resulta interesante poder corregir las anomalías encontradas.

Mi trabajo en esta línea se inició con mi incorporación al grupo ELP en enero de 2006. Podemos dividir el trabajo realizado en este periodo en dos partes: (i) enero de 2006 a enero de 2007, fecha de obtención del DEA; (ii) enero de 2007 a julio de 2007, fecha de presentación de la tesis de master completa.

Las nuevas contribuciones originales que ahora se presentan y que configuran el cuerpo de esta tesis de master son las siguientes.

Reparación semi-automática de sitios Web: (ver Capítulo 4)

Trabajo. M. Alpuente, D. Ballis, M. Falaschi, y D. Romero. *A Semi-automatic Methodology for Reparing Faulty Web Sites*. Proc. of the 4th IEEE Int'l Conference on Software Engineering and Formal Methods (SEFM'06). Pages 31 – 40. Pune, India. IEEE Computer Society Press. 2006.

Tomando como base el marco de verificación de [Alpuente et al., 2006a], se formuló una metodología para la corrección del código erróneo [Alpuente et al., 2006b]. En ella, partiendo de una categorización de los diferentes tipos de error que pueden encontrarse en un sitio Web respecto a su especificación inicial, se clasifican las distintas clases de acciones de reparación que pueden ejecutarse para corregir cada error dado. La reparación de los errores de corrección se puede resolver manual o automáticamente por medio de un *constraint solver* apropiado [Apt, 2003], el cual puede proveer valores adecuados para el término a ser insertado. En la implementación de Verdi estamos usando el *constraint solver* CiME [Projet CiME,]. De esta forma se ofrece al usuario de la herramienta un conjunto correcto y completo de acciones entre las que elegir.

La metodología de corrección se estructura entonces en dos fases. Primero se ejecutan las acciones orientadas a garantizar la corrección del sitio y, a continuación, se acometen las acciones encaminadas a recuperar la completitud. Una ventaja adicional de esta metodología es que permite detectar cualquier eventual inconsistencia entre las reglas de la especificación Web así como resolver los problemas que puedan surgir derivados de la interacción entre las acciones de reparación.

Estrategias para optimizar la reparación de la corrección (ver Sección 5.1)

Trabajo. D. Ballis y D. Romero. *Fixing Web Sites Using Correction Strategies*. Proc of 2nd Int'l Workshop on Automated Specification and Verification of Web Systems (WWV'06). Pages 495 – 502. Paphos, Cyprus. IEEE Computer Society Press. 2007.

Cómo una extensión al trabajo de reparación, en [Ballis and Romero, 2007b] se realiza un análisis sistemático de la relación entre los errores de corrección detectados por nuestra metodología y, de esta manera, se definen dos estrategias, $\mathcal{M}$*-strategy* y $\mathcal{MNO}$*-strategy*.

$\mathcal{M}$*-strategy* minimiza el número de acciones de reparación a ser ejecutadas. Lo que se pretende es reducir el esfuerzo necesario para reparar el sitio Web.

$\mathcal{MNO}$*-strategy* minimiza la información a ser cambiada o eliminada sobre el sitio Web. El objetivo es mantener la mayor similitud posible entre el sitio Web original y el sitio obtenido después de aplicar las acciones reparadoras.

En ambas estrategias, el número de errores que es necesario reparar para obtener un sitio Web correcto es menor que el número de errores originales del sitio. Como consecuencia, éstas estrategias garantizan una mejor *performance* del sistema de reparación.

Estrategias de reparación para sitios Web incompletos (ver Capítulo 5.2)

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. **(Submited)**. Notificación de aceptación: 1 de Septiembre.

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.

Análisis de información no demandada (ver Sección 6.1)

Trabajo. M. Alpuente, P. Mello, D. Romero y M. Zanella. *A framework for analysis of undemanded data in Web sites*. Technical Report DSIC-II/14/07. DSIC, UPV. Julio 10, 2007.

La metodología de verificación dada en [Alpuente et al., 2006a] identifica dos categorías de errores: información incorrecta e información faltante dentro de sitio Web. Sobre éste análisis, se investigó una tercera categoría de errores, que llamamos *undemandedness error*, en donde se considera la información innecesaria con respecto a una especificación Web. En este trabajo, se intenta detectar posibles errores que puedan ser ocasionados por información no solicitada por el usuario. De ésta investigación, se obtuvo un algoritmo capaz de: detectar posibles errores de *undemandedness*, interactuar con el usuario para resolverlos, e incorporar nuevas reglas de completitud a la especificación original.

Filtrado de documentos XML: (ver Sección 6.2)

Trabajo. D. Ballis y D. Romero. *Filtering of XML Documents*. Proc of 2nd Int'l Workshop on Automated Specification and Verification of Web Systems (WWV'06). Pages 503 – 509. Paphos, Cyprus. IEEE Computer Society Press. 2007.

En [Ballis and Romero, 2007a] exploramos una aplicación de las técnicas de reescritura parcial para el filtrado de documentos XML. Esencialmente, se define un lenguaje de especificación que permite extraer información relevante (filtrado

positivo), como así también excluir contenido no deseado (filtrado negativo) de un documento XML.

Un marco genérico abstracto para la verificación de sitios Web (ver Capítulo 7)

Trabajo. M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, y D. Romero. *An Abstract Generic Framework for Web Sites Verification.* York Doctoral Symposium (YDS 2007). October 26, 2007 – King's Manor, University of York, UK. **(Submited)**. Notificación de aceptación, 17th September 2007.

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., 2007e], 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 8)

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. June 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 apartado 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 una gran *performance* (ej. menos de un segundo para evaluar un árbol de aproximadamente 30.000 nodos). También aquí, 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 es reducir el costo de aprendizaje, motivo por el cual, tenemos desarrollada una amigable interfaz de usuario. El prototipo resultante 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 de la tesis

El resto 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 4 describimos la metodología de reparación. En el Capítulo 5, presentamos las estrategias para optimizar la reparación de la corrección y de la completitud. En el Capítulo 6 el análisis de información no demandada detectada dentro de un sitio Web y la aplicación de las técnicas desarrolladas en un lenguaje filtrado de documentos XML. En el Capítulo 7, 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 8, 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 9 describimos las conclusiones y los trabajos futuros junto con un cronograma tentativo de hacia la tesis doctoral.

2

Preliminares

En esta sección, recordamos algunas definiciones preliminares junto con la terminología necesaria para el entendimiento 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 Σ 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 Σ, respectivamente. Los términos se consideran árboles etiquetados de la manera usual.

Las posiciones son representadas por secuencias de números naturales que denotan el camino de acceso a un término. La secuencia vacía Λ 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 con-

diciones: $(i)X_i \neq X_j$, 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 ε representamos una *sustitución vacía*. Dada una sustitución σ, el *dominio* de σ es el conjunto $Dom(\sigma) = \{X | X\sigma \not\equiv X\}$. Dadas las sustituciones σ_1 y σ_2, tal que $Dom(\sigma_1) \subseteq Dom(\sigma_2)$, por σ_1/σ_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 σ 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 (Σ, R), donde Σ 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 (Σ, 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 σ 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 $\mathtt{s}^{\mathtt{n}}(\mathtt{0})$. Usamos la notación estándar para las listas, siendo [ ] la lista vacía. Los* strings *son listas de caracteres, como de costumbre.*

3

Verificación de sitios Web

En esta sección recordaremos la metodología de verificación formal propuesta en [Alpuente et al., 2006a], la cual permite detectar información prohibida como así también faltante dentro de un sitio Web. La ejecución de una especificación Web sobre un sitio Web, permite renococer y localizar la información que discrepa entre el sitio y las propiedades establecidas por la 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.

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 programas 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, estas 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 *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-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.

(a)Página Web

```
<blog>

 <owner>Daniel Romero</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)Término del Algebra

```
blog(

  owner(Daniel Romero),

  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

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* $p1$ *tenemos dos entradas del blog,* $p2$ *y* $p3$ *corresponden a las entradas de culture y history respectivamente, allí estan los comentarios de cada una de esas entradas. La página* $p4$ *es la que contiene los miembros del blog.*

p1) blog(owner(Daniel Romero),
entry(subject(culture),
date(02-12-06),
content(blablabla1),
comments(1)),
entry(subject(history),
date(01-08-06),
content(blablabla2),
comments(0)))

p2) entry(subject(culture),
date(02-12-06),
comments(commentary(date(02-15-06),
author(blink(Daniel Romero)),
text(bla SEX bla))),
commentary(date(06-23-06),
author(Alan Turing),
text(blabla)))

p3) entry(subject(history),
date(01-08-06),
comments())

p4) members(member(name(Daniel Romero),age(32)),
member(name(Demis Ballis), age(31)),
member(name(Peter Pan), age(14)))

Figura 3.2: Ejemplo de un sitio Web

3.2. Lenguaje de especificación

En [Alpuente et al., 2006a], una especificación Web es una triupla (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 es formalizado como un sistema de reescritura de términos el cual es tratado como 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: $\texttt{l} \rightharpoonup \texttt{error} \mid$

$\mathtt{C}$, 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. $\mathtt{X} \in \mathtt{rexp}$) w.r.t. una 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 corresponde 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 σ, si (i) cada estructura de texto $\mathtt{X_i}\sigma$, $\mathtt{i} = 1, \ldots, \mathtt{n}$, está contenido 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 $\mathtt{p}$ es considerada incorrecta si una instancia $\mathtt{l}\sigma$ de $\mathtt{l}$ es *reconocida* dentro de $\mathtt{p}$ y $\mathtt{C}$ se cumple para σ.

El tercer conjunto de reglas I_M permite especificar propiedades para detectar páginas Web incompletas o faltantes (*reglas de completitud*). Una regla de completitud es definida como $\mathtt{l} \rightharpoonup \mathtt{r}\ \langle \mathtt{q} \rangle$, donde $\mathtt{l}$ y $\mathtt{r}$ 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, las que deberán estar definidas en R. Intuitivamente, la interpretación de una regla universal $\mathtt{l} \rightharpoonup \mathtt{r}\langle \mathtt{A} \rangle$ (respectivamente, una regla existencial $\mathtt{l} \rightharpoonup \mathtt{r}\langle \mathtt{E} \rangle$) w.r.t. un sitio Web W es el siguiente: si (una instancia de) $\mathtt{l}$ es reconocida en W, también (una instancia de) la forma irreducible de $\mathtt{r}$ debe ser reconocida en *todas* (respectivamente, *algunas*) páginas Web las cuales embeban (una instancia de) $\mathtt{r}$.

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 es usada para seleccionar el subconjunto de páginas Web del sitio sobre el cual vamos a verificar la propiedad. Más específicamente, la regla r es ejecutada sobres las páginas que embeban la información marcada.

[1] El lenguaje regular es representado por medio de la usual sintaxis de expresiones regulares

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

Figura 3.3: Ejemplo de una especificación Web

Ejemplo 3.2.1 *Considere el sitio Web de la Figura 3.2, y la especificación Web de la Figura 3.3. Luego tenemos. 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 comentarios 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 dueño 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 es llevada a cabo por la ejecución de la especificación Web sobre el sitio Web. Esto es mecanizado por medio de *reescritura parcial*, un técnica de reescritura novedosa que se obtiene remplazando el *patter-matching* tradicional de 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" desde una página, y luego junta las piezas para reescribirlas un nuevo término. El armado es

realizado por medio de árboles de simulación, 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 es ignorada 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:

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

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.

A continuación se dará la noción de reescritura parcial, en este caso se deja de lado las 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* σ *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.

[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.

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 desde una página Web $\mathtt{s}$, una subparte de $\mathtt{s}$ la cual es simulada por una instancia *ground* de $\mathtt{l}$, y remplazar $\mathtt{s}$ por una instancia *ground* reducida de $\mathtt{r}$. Note que en el contexto de la expresión reducida seleccionada $\mathtt{s}_{|u}$ es dejada de lado después de cada paso de reescritura. Por notación $\mathtt{s} \rightharpoonup_I \mathtt{t}$, denota que $\mathtt{s}$ es parcialmente reescrito 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 w.r.t. 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 w.r.t. una especificación Web.

A continuación se describe como son detectados éstos errores de corrección y completitud.

3.4.1. Detectando errores de corrección

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

Mas precisamente, dada una página Web $\mathtt{p}$ y una regla de corrección $\mathtt{l} \rightharpoonup \mathtt{error} \mid \mathtt{C}$, primero se intenta reconocer $\mathtt{l}$ dentro $\mathtt{p}$ por medio de reescritura parcial. Luego son analizandos los valores tomados por las variables de $\mathtt{C}$, los cuales son obtenidos del paso reescritura parcial. Si el texto estructurado, el cual está limitado a cada variables de $\mathtt{C}$, pertenece al lenguaje de la correspondiente expresión regular y se cumplen todas las ecuaciones instanciadas en $\mathtt{C}$, el fallo de

la página Web $\mathtt{p}$ y la información del error detectado son dados al usuario. Esta metodología permite precisar cual 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 $\mathtt{l}$ de una regla de corrección que pertenece a I_N tal que $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$.*

Dado un error de corrección $e = (\mathtt{p}, w, \mathtt{l}, \sigma)$, $\mathtt{p}$ es la página donde se produce el error, w es la posición del error dentro de la página, $\mathtt{l}$ es la parte izquierda de la regla que produzco 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 w.r.t. un conjunto de reglas de corrección I_N. Cuando no exista confusión, simplemente se escribirá E_N.

Ejemplo 3.4.2 *Considere el sitio Web de la Figura 3.2, y la especificación de la Figura 3.3. Los siguientes errores de corrección son detectados:*

- *La página $p2$ del sitio contiene la sentencia $blink$, la cual está prohibida por la regla de corrección r_1 de la especificación*
  $e_{N1} = (p2, 3.1.2.1, blink(X), \{X/Daniel\ Romero\})$.

- *La página $p2$ del sitio, en el texto del comentario tenemos $blaSEXbla$, y mirando la regla r_2 tenemos que la palabra SEX no está permitida dentro de los mismos*
  $e_{N2} = (p2, 3.1.3, text(X), \{X/bla\ SEX\ bla\})$.

- *La página $p4$ dice que $PeterPan$ tiene 14 años, y por la regla r_3 tenmos que la edad mínima es de 18 años*
  $e_{N3} = (p4, 3.2, member(age(X)), \{X/14\})$.

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

3.4.2. Detectando errores de completitud

Esencialmente, la idea de detectar errores de completitud es computar el conjunto de todas las posibles expresiones marcadas que puedan ser derivadas desde el sitio Web W via las reglas de completitud de una especificación Web por medio de reescritura parcial. Estos términos marcados pueden ser pensados como requerimientos a ser satisfecho por W. Entonces, el chequeo de errores es realizado si los requerimientos son satisfechos por W usando simulación, la información del marcado y los cuantificadores. Resumiendo, el método trabaja en dos pasos:

1. Computar el conjunto de requerimientos $\mathtt{Req}_{\mathtt{M,W}}$ para W w.r.t. $I_{\mathtt{M}}$;

2. Chequear $\mathtt{Req}_{\mathtt{M,W}}$ 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 es llamado *universal* si $\mathtt{q} = \mathtt{A}$, mientras que es llamando *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. A continuación daremos la definición de cada uno de ellos.

Definición 3.4.3 (página Web faltante) *Sean W un sitio Web y (I_M, I_N, R) una especificación Web. Entonces, el par $(\mathtt{r}, W)$ es un error de página web* faltante *si exsite $\mathtt{p} \in W$ s.t. $\mathtt{p} \rightharpoonup_{I_M}^{+} \mathtt{r}$ y $\mathtt{r} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ no pertenece a W.*

La Definición 3.4.3 establece que, cuando un error de página Web faltante $(\mathtt{r}, W)$ es detectado, la expresión $\mathtt{r}$ no aparece en ninguna parte del sitio Web W.

Con el objetivo de formalizar los errores universales como así también los existenciales, se introduce la siguiente definición.

Definición 3.4.4 *Sean $\mathtt{P}$ un conjunto de términos en $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$ y $\mathtt{r} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$. Se dice que $\mathtt{P}$ es* universalmente *(resp.* existencialmente*)* completo *w.r.t. $\mathtt{r}$ sii para cada $\mathtt{p} \in \mathtt{P}$ (resp. para algina $\mathtt{p} \in \mathtt{P}$), existe $w \in O_{\mathcal{T}ag}(\mathtt{p})$ s.t. $\mathtt{r} \trianglelefteq \mathtt{p}_{|w}$.*

Definición 3.4.5 (error de completitud universal) *Sean W un sitio Web, y (I_M, I_N, R) una especificación Web. Entonces, la triupla $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{A})$ es un* error de completitud universal, *si existe $\mathtt{p} \in W$ s.t. $\mathtt{p} \rightharpoonup^{+}_{I_M} \mathtt{r}$ y $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ no es universalmente completo w.r.t. $\mathtt{r}$, $\mathtt{p}_i \in W$, $i = 1, \ldots, n$.*

Definición 3.4.6 (error de completitud existencial) *Sean W un sitio Web y (I_M, I_N, R) una especificación Web. Entonces, la triupla $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{E})$ es un* error de completitud existencial, *si existe $\mathtt{p} \in W$ s.t. $\mathtt{p} \rightharpoonup^{+}_{I_M} \mathtt{r}$ y $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ no es existencialmente completo w.r.t. $\mathtt{r}$, $\mathtt{p}_i \in W$, $i = 1, \ldots, n$.*

La Definición 3.4.5 (resp. la Definición 3.4.6) formaliza el hecho de que un requerimiento debe ocurrir en todas (resp. algunas de) las páginas Web de un subconjunto de W.

Ejemplo 3.4.7 *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
  $e_{M1} = (r_5, \{p4\}, E)$.
- *Por la regla de completitud r_6, debería existir una página que sea la* home page *del dueño del blog, por lo tanto, tenemos el* error de página Web faltante
  $e_{M2} = (r_6, W)$.

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

La metodología de verificación de [Alpuente et al., 2006a] genera los conjuntos de errores de corrección y completitud E_N y E_M mencionados para un sitio Web w.r.t. una especificación Web. Usando estos conjuntos, en la siguiente sección se formula el método presentado en [Alpuente et al., 2006b] para reparar los errores y permitir entregar un sitio Web que sea *correcto* y *completo* w.r.t. su especificación Web.

Definición 3.4.8 (sitio Web correcto) *Dada una especificación Web* (I_M, I_N, R)*, un sitio Web* W *es* correcto *w.r.t.* (I_M, I_N, R) *sii el conjunto* E_N *de errores de corrección w.r.t.* I_N *es vacío.*

Definición 3.4.9 (sitio Web completo) *Dada una especificación Web* (I_M, I_N, R)*, un sitio Web* W *es* completo *w.r.t.* (I_M, I_N, R) *sii el conjunto* E_M *de errores de completitud w.r.t.* I_N *es vacío.*

4

Reparación semi-automática de sitios Web

En este capítulo, se describe la metodología de reparación semi-automática de sitios Web presentada en [Alpuente et al., 2006b].

Dado un sitio Web W y los conjuntos de errores $\mathtt{E}_N$ y $\mathtt{E}_M$, correspondiente a W w.r.t. una especificación Web, el objeto es modificar el sitio Web agregando, cambiando o eliminado la información errónea para producir un sitio Web que sea correcto y completo w.r.t. la especificación Web considerada. Para este propósito y, en correspondencia con la categorización de errores distinguidas en 3.4.1 y 3.4.2, se introduce un catálogo de *acciones reparadoras* que pueden ser aplicadas al sitio Web para reparar los errores. De esta manera, en esta metodología, reparar un sitio Web consiste en seleccionar un conjunto adecuado de acciones reparadoras que serán ejecutadas para eliminar la información errónea del sitio.

En primer lugar veremos las diferentes acciones reparadoras. Luego, se presentan las estrategias definidas para los errores de corrección y completitud respectivamente. Por último, daremos una aproximación que permite, bajo ciertas condiciones, reparar de forma automática algunos de los errores detectados en un sitio Web.

4.1. Acciones reparadoras

Con el objetivo de reparar un sitio Web, son definidas cuatro acciones reparadoras. Las acciones consideradas son las siguientes:

- **change**$(\mathtt{p}, w, \mathtt{t})$ remplaza el subtérmino $\mathtt{p}|_w$ en $\mathtt{p}$ con el término t $\mathtt{t}$. Retorna el término $\mathtt{p}$ modificado.
- **insert**$(\mathtt{p}, w, \mathtt{t})$ modifica el término $\mathtt{p}$ agregando el término $\mathtt{t}$ en $\mathtt{p}_{|w}$.
- **add**$(\mathtt{p}, W)$ agrega la página Web $\mathtt{p}$ al sitio Web W.
- **delete**$(\mathtt{p}, t)$ borra todas las ocurrencias del término $\mathtt{t}$ en la página Web $\mathtt{p}$.

Estas acciones serán usadas como primitivas en las estrategias de reparación que se presentan más adelante.

Note que es posible que un error particular pueda ser reparado por medio de diferentes acciones. Por ejemplo, un error de corrección puede ser reparado borrando la información incorrecta/prohibida, o cambiando el dato que produjo el error. Del mismo modo, un error de completitud puede ser reparado insertando la información faltante o borrando de todo el sitio Web los datos que causaron el error. Por otra parte, modificar o insertar información de manera arbitraria puede causar que aparezcan nuevos errores de corrección. Por ello es necesario asegurar que el dato ingresado/modificado sea seguro (*safe*) w.r.t. una especificación Web. Para este propósito se introduce la siguiente definición.

Definición 4.1.1 (safe) *Sean* (I_M, I_N, R) *una especificación Web and* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Entonces,* $\mathtt{p}$ *es* safe *w.r.t.* I_N*, sii para cada* $w \in O_{\mathcal{T}ag}(\mathtt{p})$ *y* $(\mathtt{l} \rightharpoonup \mathtt{r} \mid \mathtt{C}) \in I_N$

- *no existe un* σ *s.t.* $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$*; ó*
- $\mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w}$*, pero no se cumple* $\mathtt{C}\sigma$*.*

A continuación, se describe la metodología de reparación que permite corregir los errores de corrección y de completitud. Esta metodología trabaja en dos etapas. En la primer etapa, se infieren y ejecutan las acciones reparadores correspondiente a los errores de corrección, de esta manera se obtiene un sitio Web que sea correcto, pero que aún pueda ser incompleto. En la segunda etapa, se hace lo propio sobre los errores de completitud, obteniendo así un sitio Web que sea correcto y completo w.r.t una especificación Web.

4.2. Reparando errores de corrección

El procedimiento de reparar los errores de corrección es el siguiente: siempre que un error de corrección es detectado, se elije una posible acción reparadora (según las diferentes acciones descritas en 4.1) que será ejecuta para eliminar la información errónea, siempre que la ejecución de la misma no produzca un nuevo error.

En esta sección se considerará a un sitio Web W, a una especificación Web (I_M, I_N, R), y a el conjunto de errores de corrección $E_N \neq \emptyset$ w.r.t. I_N para W.

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, desde $\mathtt{p}_{|w}$), (ii) cambiando $\mathtt{l}\sigma$ por un término que sea correcto; por lo tanto, es posible elegir entre las siguientes estrategias reparadoras:

- Corrección por eliminación
- Corrección por cambio

4.2.1. Estrategia de corrección por eliminación

Este caso, simplemente es eliminar todas la ocurrencias del subtérmino $\mathtt{p}_{|w}$ de la página Web $\mathtt{p}$ que contiene la información erronea $\mathtt{l}\sigma$, aplicando la acción reparadora **delete**$(\mathtt{p}, \mathtt{p}_{|w})$.[1]

Ejemplo 4.2.1 *Sea $e_{M1} = (p2, 3.1.2.1, blink(X), \{X/Daniel\ Romero\})$ el error de corrección presentado en el Ejemplo 3.4.2. Aplicando la estrategia de corrección por eliminación se obtiene lo siguiente:*

$p2' =$ ***delete***$(p2, p2_{3.1.2.1})$
donde
p2′ = entry(subject(culture), date(02-12-06),
comments(commentary(date(02-15-06),
author(),

[1]Note que, en lugar de eliminar todo el subtérmino $\mathtt{p}_{|w}$, otra opción más precisa, pero también con un costo 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.

text(bla SEX bla))),
commentary(date(06-23-06),
author(Alan Turing),
text(blabla)))

4.2.2. Estrategia de corrección por cambio

Dado un error de corrección $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$, el proceso es remplazar el subtérmino $\mathtt{p}_{|w}$ de la página Web $\mathtt{p}$ con un nuevo término t introducido por el usuario.

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. En este sentido, se define la siguiente *propiedad de corrección global* que impide que un nuevo término t produzca un nuevo error al ser insertado en una página $\mathtt{p}$.

En primer lugar daremos la definición de una función auxiliar.

Definición 4.2.2 *Sean* $\mathtt{s}, \mathtt{t} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *s.t.* $\mathtt{s} \trianglelefteq \mathtt{t}$. *Se define al conjunto* $Emb_{\mathtt{s}}(\mathtt{t})$ *como el conjunto de todas las posiciones en* $\mathtt{t}$ *que embeben algún subtérmino de* $\mathtt{s}$. *Por ejemplo, considere los términos* $f(k, g(c))$, *y* $f(b, g(c), k)$. *Entonces,* $f(k, g(c)) \trianglelefteq f(b, g(c), k)$ *y* $Emb_{f(k,g(c))}(f(b, g(c), k)) = \{\Lambda, 2, 2{,}1, 3\}$.

Definición 4.2.3 (propiedad de corrección global) *Sean* (I_M, I_N, R) *una especificación Web,* $\mathtt{p}' \equiv \mathbf{change}(\mathtt{p}, w, t)$ *será una acción reparadora que produce la página Web* $\mathtt{p}'$. *Entonces,* $\mathbf{change}(\mathtt{p}, w, t)$ obedece la propiedad de corrección global *si, para cada error de corrección* $\mathtt{e} = (\mathtt{p}', w', \mathtt{l}, \sigma)$ *w.r.t.* I_N *s.t.* $w' \leq w$,

$$\{w\}.O_{\mathcal{T}ag}(t) \cap \{w'\}.Emb_{\mathtt{l}\sigma}(\mathtt{p}'_{|w'}) = \emptyset$$

La idea en la definición 4.2.3 es que cualquier error $\mathtt{e}$ en la nueva página $\mathtt{p}' \equiv \mathbf{change}(\mathtt{p}, w, t)$, obtenida por insertar el término t en $\mathtt{p}$, no sea consecuencia de este cambio sino que el mismo hubiera existido previamente. Por este propósito, se requiere que (el conjunto de posiciones de) la información errónea $\mathtt{l}\sigma$ no se solape con el término t.

Ejemplo 4.2.4 *Sean W el sitio Web de la Figura 3.2, E la especificación Web de la Figura 3.3 para W, y $E_N = \{e_{N1}, e_{N2}, e_{N3}\}$ el conjunto de errores de corrección obtenidos en el Ejemplo 3.4.2. Sea $t = link(my\ favorite\ page)$. Al aplicar la estrategia* corrección por cambio*, para reparar el error e_{N2} usando el término t, hacemos el siguiente análisis:*

El término t no contiene errores de corrección w.r.t. E. Pero la aplicación de la acción de tipo change *nos daría*

$p2' \equiv change(p2, 3.1.3, link(my\ favorite\ page))$, *donde quedaría*
$p2' \equiv$ *entry(subject(culture), date(02-12-06),*
comments(commentary(date(02-15-06),
author(blink(Daniel Romero)),
text(link(my favority page)) *)),*
commentary(date(06-23-06),
author(Alan Turing),
text(blabla)))

observamos que, al aplicar la acción de tipo change*, se genera un nuevo error (ver la regla r_4 de E)*

$$e_N = (p2', 3.1.3, text(link(X)), \{X/my\ favorite\ page\})$$

y, por la Definición 4.2.3, tenemos que

$$Emb_{1\sigma}(p2'_{|w'}) = Emb_{text(link(my\ favority\ page))}(p2'_{|3.1.3}) = \{\Lambda\}$$

$$O_{\mathcal{T}ag}(t) = O_{\mathcal{T}ag}(link(my\ favority\ page)) = \{\Lambda\}$$

$$\{w\}.O_{\mathcal{T}ag}(t) \cap \{w'\}.Emb_{1\sigma}(p2'_{|w'}) =$$

$$\{w\}.\{\Lambda\} \cap \{w'\}.\{\Lambda\} = \{3.1.3\} \cap \{3.1.3\} = \{3.1.3\} \neq \emptyset$$

por lo tanto, la acción de tipo change *no se puede aplicar debido a que no se cumple la propiedad de "corrección global" (ver Definición 4.2.3).*

El uso de las estrategias de *corrección por eliminación* y/o *corrección por cambio* decrementa el número de errores de corrección del sitio web original, como lo establece la siguiente proposición.

Proposición 4.2.5 *Sean* (I_M, I_N, R) *una especificación Web y* W *un sitio web. Sea* $E_N(W)$ *el conjunto de errores de corrección w.r.t.* I_N *de* W*, y* $(\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N(W)$*. Por ejecución de la acción reparadora de tipo* $\mathbf{delete}(\mathtt{p}, \mathtt{p}_{|w})$ *(resp. de tipo* $\mathbf{change}(\mathtt{p}, w, t)$*) respetando la estrategia de corrección por eliminación (resp. corrección por cambio), tenemos que*

$$|E_N(W')| < |E_N(W)|$$

donde

$$W' \equiv W \setminus \{\mathtt{p}\} \cup \{\mathbf{delete}(\mathtt{p}, \mathtt{p}_{|w})\}$$

(resp. $W' \equiv W \setminus \{\mathtt{p}\} \cup \{\mathbf{change}(\mathtt{p}, w, t)\}$*)*

4.3. Reparando errores de completitud

En esta sección se explican las estrategias 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 w.r.t. una especificación Web (I_M, I_N, R). Este supuesto permite diseñar una metodología reparadora la cual "completa" el sitio Web sin introducir información incorrecta.

Sea $E_M(W)$ el conjunto de errores detectados por I_M para el sitio Web W. Al igual que en la sección anterior, cualquier error que pertenezca a $E_M(W)$ puede ser reparado agregando la información necesaria o eliminando el dato que lo produjo. De igual forma, aseguraremos que la acción que se ejecute no sea generadora de un nuevo error de correctitud/completiud para garantizar de esta manera la terminación y la validez de la metodología.

Las estrategias de reparación de errores de completitud son::

- Completitud por inserción
- Completitud por eliminación

4.3.1. Estrategia de completitud por inserción

Acorde al 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* será usada

cuando tengamos un error de página Web faltante y la acción *insert* cuando el error de completitud sea universal o existencial.

Error de página Web faltante. Sea $e = (\mathtt{r}, W)$ un error de página Web faltante, e puede ser reparado agregando al sitio Web W una página Web $\mathtt{p}$ que embeba la expresión $\mathtt{r}$. De esta manera tenemos

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

Error de completitud existencial. Sea $e = (\mathtt{r}, \{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n\}, \mathtt{E})$ un error de completitud existencial, repararemos el error insertando un término t que embeba la expresión $\mathtt{r}$, en una página arbitraria $\mathtt{p}_i$, $i = 1, \ldots, n$. El usuario deberá proveer la posición w dentro de $\mathtt{p}_i$ donde el término t será insertado. De esta manera tenemos

$$W = W \setminus \{\mathtt{p}_i\} \cup \{\mathbf{insert}(\mathtt{p}_i, w, t)\}, \text{ donde } \mathtt{r} \trianglelefteq \mathtt{p}_{i|w} \text{ para algún } w \in O_{\mathcal{T}ag}(\mathtt{p}).$$

Error de completitud universal. Sea $e = (\mathtt{r}, \{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n\}, \mathtt{A})$ un error de completitud universal, repararemos el error insertando un término t_i que embeba la expresión $\mathtt{r}$ en cada página Web $\mathtt{p}_i$, $i = 1, \ldots, n$ que no embeba $\mathtt{r}$. El usuario deberá proveer cada posición w_i en $\mathtt{p}_i$ donde se insertará t_i De esta manera, el sitio Web W será transformado como sigue

Para cada $\mathtt{p}_i \in \{\mathtt{p}_1, \mathtt{p}_2, \ldots, \mathtt{p}_n\}$ s.t. $\mathtt{r} \not\trianglelefteq \mathtt{p}_{i|w_j}$,
  Para cada $w_j \in O_{\mathcal{T}ag}(\mathtt{p}_i)$,
    $W = W \setminus \{\mathtt{p}_i\} \cup \{\mathbf{insert}(\mathtt{p}_i, w_i, t_i)\}$,
    donde $\mathtt{r} \trianglelefteq \mathtt{p}_{|w_i}$ para algún $w_i \in O_{\mathcal{T}ag}(\mathtt{p}_i)$.

En ambas acciones, **add** e **insert**, se introduce nueva información en el sitio Web que puede ser potencialmente dañina, es decir, que puede contener información errónea o incompleta. Es importante restringir el tipo de información que el usuario puede ingresar y obligar a que ésta sea segura. Por lo tanto, la nueva información agregada por alguna acción reparadora no introducirá un nuevo error de corrección

Proposición 4.3.1 *Sean* (I_M, I_N, R) *una especificación Web y* W *un sitio Web correcto w.r.t.* (I_M, I_N, R). *Sean* $\mathtt{p}_1 \equiv \mathbf{insert}(\mathtt{p}, w, t)$ *y* $\mathtt{p}_2 \equiv \mathbf{add}(\mathtt{p}_2, W)$.

- *Si* $\mathtt{p}_1$ *es* safe *w.r.t.* I_N, *entonces* $W \setminus \{\mathtt{p}\} \cup \{\mathtt{p}_1\}$ *es correcto w.r.t.* (I_M, I_N, R).
- *Si* $\mathtt{p}_2$ *es* safe *w.r.t.* I_N, *entonces* $W \cup \{\mathtt{p}_2\}$ *es correcto w.r.t.* (I_M, I_N, R).

También se busca prevenir que la ejecución de la acción reparadora no introduzca un nuevo error de completitud, es decir, el conjunto de errores que se repare sea solamente el conjunto inicial, que llamaremos $E_M(W)$. Dado un error de completitud $(\mathtt{e}, \mathtt{W})$, se usará la notación $\mathtt{e}(\mathtt{r})$ para hacer evidente el requerimiento insatisfecho $\mathtt{r}$ por el error $\mathtt{e}$.

Definición 4.3.2 (acceptable) *Sean* (I_M, I_N, R) *una especificación Web y* W *un sitio Web w.r.t.* (I_M, I_N, R). *Sea* $E_M(W)$ *el conjunto de errores de completitud de* W *w.r.t.* I_M.

- *la acción reparadora* $\mathtt{p}_1 \equiv \mathbf{insert}(\mathtt{p}, w, t)$ *es* acceptable *w.r.t.* (I_M, I_N, R) *y* W *sii*

  1. $\mathtt{p}_1$ *es* safe *w.r.t.* (I_M, I_N, R);
  2. $\mathtt{r} \trianglelefteq t_{|w}$, $w \in O_{\mathcal{T}ag}(t)$, *para algún* $\mathtt{e}(\mathtt{r}) \in E_M(W)$;
  3. *si* $W' \equiv W \setminus \{\mathtt{p}\} \cup \{\mathtt{p}_1\}$, *entonces* $E_M(W') \subset E_M(W)$.

- *la acción reparadora* $\mathtt{p}_2 \equiv \mathbf{add}(\mathtt{p}_2, W)$ *es* acceptable *w.r.t.* (I_M, I_N, R) *y* W *sii*

  1. $\mathtt{p}_2$ *es* safe *w.r.t.* (I_M, I_N, R);
  2. $\mathtt{r} \trianglelefteq \mathtt{p}_{2|w}$, $w \in O_{\mathcal{T}ag}(\mathtt{p}_2)$, *para algún* $\mathtt{e}(\mathtt{r}) \in E_M(W)$;
  3. *si* $W' \equiv W \cup \{\mathtt{p}_2\}$, *entonces* $E_M(W') \subset E_M(W)$.

La Definición 4.3.2 garantiza que la información que sea agregada por la acciones **add** o **insert** sea correcta y no se genere un nuevo error de completitud. Más precisamente, el número de errores de completitud decrece por el efecto de la ejecución de tales acciones reparadoreas.

4.3.2. Estrategia de completitud por eliminación

En algunas situaciones es más conveniente eliminar la información incompleta; en particular, esta opción puede ser muy usada cuando se tiene información desactualizada. La principal idea de la estrategia de eliminación es eliminar toda la información del sitio Web que causó el error de completitud. La estrategia es independiente del tipo de error que estemos analizando; por lo tanto, la información faltante es computada de la misma manera para los tres tipos de errores. En otras palabras, dada la expresión faltante $\mathtt{r}$ de un error de completitud $\mathtt{e}(\mathtt{r})$ (esto es, un error de página faltante $(\mathtt{r}, W)$, o un error de completitud existencial $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{E})$, o un error de completitud universal $(\mathtt{r}, \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, \mathtt{A})$), existe una página Web $\mathtt{p} \in W$ tal que $\mathtt{p} \rightharpoonup^+ \mathtt{r}$. Por lo tanto, el proceso computa y elimina de las páginas Web todos los términos ocurridos en la secuencia de reescritura parcial que se obtienen de la expresión faltante $\mathtt{r}$. Este problema es generalmente indecidible, por ello algunas restricciones sobre el lenguaje de especificación debieron ser consideradas (para más detalle, ver la especificación de *bounded* en [Ballis, 2005]).

Mas formalmente, dada una especificación Web (I_M, I_N, R), un sitio Web W y un error de completitud $\mathtt{e}(\mathtt{r})$, el sitio Web W cambiará de la siguiente manera.

Para cada $t_1 \rightharpoonup t_2 \rightharpoonup \ldots \rightharpoonup \mathtt{r}$, donde $t_1 \trianglelefteq \mathtt{p}_{|w}$, $w \in O_{\mathcal{T}ag}(\mathtt{p})$, $\mathtt{p} \in W$

$$\begin{aligned} W \equiv &\{\mathtt{p} \in W \mid t_i \not\trianglelefteq \mathtt{p}_{|w}, \forall\, w \in O_{\mathcal{T}ag}(\mathtt{p}), i = 1, \ldots, n\} \cup \\ &\{\mathbf{delete}(\mathtt{p}, t_i) \mid \mathtt{p} \in W, t_i \trianglelefteq \mathtt{p}_{|w}, w \in O_{\mathcal{T}ag}(\mathtt{p}) \\ &i = 1, \ldots, n\} \end{aligned}$$

4.4. Reparación automática de errores

La aplicación de las estrategias vistas hasta ahora permiten garantizar la terminación de esta metodología de reparación, como así también, que la ejecución de las acciones reparadoras no producirán efectos no deseados en el sitio w.r.t. una especificación Web. Pero en todos éstos casos es necesaria la intervención del usuario, debido a que es éste quien debe proveer el término t a ser agregado o, en su defecto, decidir la eliminación de la información errónea. Pero un caso particular a tener presente es cuando en los errores de corrección la regla que

detecta el error es condicional, es decir, tiene la siguiente forma

$$\mathtt{l} \rightharpoonup \mathtt{error} \mid \mathtt{C}, \text{ con } \mathtt{C} \neq \emptyset \text{ y } Var(\mathtt{C}) \subseteq Var(\mathtt{l})$$

en ésta situación, es posible encontrar de forma automática un término $t = \mathtt{l}\sigma$ que podría ser utilizado en la estrategia de corrección por cambio y, de esta manera, automatizar la metodología evitando la participación del usuario en el proceso. A continuación se presenta cómo obtener la substitución σ y el término t.

Considere el error de corrección $\mathtt{e} = (\mathtt{p}, w, \mathtt{l}, \sigma) \in E_N$, donde $\mathtt{l}$ pertenece a la parte izquierda de una regla de corrección condicional, la acción reparadora **change**$(\mathtt{p}, w, t)$, y el siguiente conjunto de condiciones

$$CS_{\mathtt{e}} \equiv \{\neg\mathtt{C} \mid \exists \quad (\mathtt{l} \rightharpoonup \mathtt{r} \mid \mathtt{C}) \in I_N, \text{ una posición } w', \\ \text{y una substitución } \sigma \text{ s.t. } \mathtt{l}\sigma \trianglelefteq \mathtt{p}_{|w.w'}\}$$

Llamaremos a $CS_{\mathtt{e}}$ *problema de satisfacción de restricciones* asociado al error $\mathtt{e}$. Este conjunto se obtiene negando y recopilando todas las condiciones de aquellas reglas de corrección en donde su parte izquierda embeba a $\mathtt{p}_{|w}$. Tal conjunto de restricciones, que puede ser resuelto manualmente o de manera automática por medio de un *constraint solver* apropiado [Apt, 2003], puede ser utilizado para obtener valores convenientes para formar un nuevo término t a ser insertado.

Denotamos como $Sol(CS_{\mathtt{e}})$ al conjunto de todas las asignaciones que verifican las restricciones en $CS_{\mathtt{e}}$. La restricción de $Sol(CS_{\mathtt{e}})$ a las variables ocurridas en σ es denotada por $Sol(CS_{\mathtt{e}})_{|\sigma}$.

A continuación se presenta un ejemplo que ilustra la situación descrita.

Ejemplo 4.4.1 *Sean W el sitio Web de la Figura 3.2, E la especificación Web de la Figura 3.3 para W, y $E_N = \{e_{N1}, e_{N2}, e_{N3}\}$ el conjunto de errores de corrección obtenidos en el Ejemplo 3.4.2. El error $e_{N3} = (p4, 3.2, member(age(X)), \{X/14\})$. es detectado por la regla condicional r_3, de esta manera tenemos*

$$CS_{\mathtt{eN3}} \equiv \{\neg(X < 18)\} \textit{ donde } \sigma = \{X/14\}$$

sea $\sigma' = \{X/21\}$ una solución para $CS_{\mathtt{eN3}}$ aportada por un constraint solver*, luego*

$$\{X/21\} \in Sol(CS_{\mathtt{eN3}})_{|\sigma'}$$

obteniendo de esta manera los valores apropiados para formar el término t, y poder aplicar la acción de tipo change *de la siguiente forma*

$$t = l\sigma' = member(age(21))$$

$$change(p4, 3.2, member(age(21)))$$

reparando así el error de corrección sin la participación del usuario.

4.4.1. Conflicto entre reglas

Algunos sistemas de restricciones pueden no tener solución, esto significa que existen dos o más reglas que se encuentran en conflicto. Consideremos estas dos situaciones

a) Dado el conjunto de reglas de corrección

$$I_N = \{l \rightharpoonup \texttt{error} \mid c, l \rightharpoonup \texttt{error} \mid \neg c\}$$

tendríamos el conjunto sin solución $CS_{\texttt{e}} = \{\neg c, c\}$.

b) Dado el cojunto de reglas de corrección

$$I_N = \{l_1 \rightharpoonup \texttt{error}, l_2 \rightharpoonup \texttt{error} \mid c\} \text{ donde } l_1 \trianglelefteq l_2$$

obtendríamos un $CS_{\texttt{e}} = \{\neg c\}$ y $t = l_2\sigma$ (siendo σ una solución de $CS_{\texttt{e}}$). Pero también tendríamos que $l_1 \trianglelefteq t$; por lo tanto, t no cumpliría con la Definición 4.2.3.

En ambas situaciones, para dar una solución, es necesario pedirle al usuario que repare la especificación Web antes de proceder.

5

Estrategias de optimzación

En este capítulo se presentan dos trabajos que son derivados de la metodología de reparación, formulada en el Capítulo 4.

En primer lugar, proponemos dos estrategias que optimizan el proceso de reparación de sitios Web incorrectos [Ballis and Romero, 2007b]. A continuación, se presenta como obtener un sitio Web correcto y completo por medio de estrategias de reparación para sitios Web incompletos [Alpuente et al., 2007d].

5.1. Estrategias para optimizar la reparación de la corrección

La metodología de verificación del Capítulo 3 permite reconocer de forma automática información errónea dentro de un sitio Web w.r.t. una especificación Web. Una extensión a ésta metodología, se introdujo en el Capítulo 4. Allí vimos como reparar los errores detectados por medio de la ejecución de una secuencia de acciones reparadoras que son inferidadas semi-automáticamente por el sistema.

Cómo una optimización de la metodología de reparación, se analiza la dependencia existente entre los errores de corrección y, explotando ese conceptos, se formalizan dos estrategias de corrección [Ballis and Romero, 2007b]. Por un lado, $\mathcal{M}$-*strategy* permite minimizar el número de acciones reparadoras a ser ejecutadas, mientras que $\mathcal{MNO}$-*strategy* reduce la cantidad de información a ser cambiada/eliminada del sitio Web. En ambas estrategias, el número de errores necesarios a ser reparados para obtener un sitio Web correcto es menor que el número de errores correspondiente a W. Como consecuencia, estas estrategias garantizan una mejor *performance* del sistema de reparación.

A continuación se describe el análisis de las dependencia de los errores de corrección y se presentan las dos estrategias de corrección.

5.1.1. Dependencia entre errores de corrección

Típicamente, una página Web puede contener varios errores de corrección los cuales pueden estar de alguna manera conectados. Cómo la ejecución de una acción reparadora puede arreglar más de un error, es importante detectar si un error depende de otro. A continuación analizaremos la dependencia entre errores de corrección.

En primer lugar definiremos la noción de orden entre errores, la cual es inducida por la posición del error dentro de la página Web.

Definición 5.1.1 *Sean* $e_1 \equiv (\mathtt{p}, w_1, \mathtt{l}_1, \sigma_1, C_1)$ *y* $e_2 \equiv (\mathtt{p}, w_2, \mathtt{l}_2, \sigma_2, C_2)$ *dos errores de corrección en* $\mathbb{E}_N(\mathtt{p})$*. Entonces,* $e_1 \preceq e_2$ *sii* $w_1 \leq w_2$*.*

Decimos que e_1 y e_2 son no comparables (w.r.t. $\preceq$) sii $e_1 \not\preceq e_2$ y $e_2 \not\preceq e_1$.

De la Definición 5.1.1 se puede obtener el siguiente resultado.

Proposición 5.1.2 *Sean* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web, y* $e_i = (\mathtt{p}, w_i, \mathtt{l}_i, \sigma_i, C_i) \in \mathbb{E}_N(\mathtt{p})$*,* $i = 1, \ldots, n$*, tal que* $e_1 \preceq e_2 \preceq \ldots \preceq e_n$*. Entonces tenemos:*

- *Si* $p' \equiv \mathbf{change}(\mathtt{p}, w_1, t)$ *es una acción reparadora* safe*, entonces* $p'_{|w_1} \equiv t$ *está reparada.*
- *Si* $p' \equiv \mathbf{delete}(\mathtt{p}, w_1, t)$ *es una acción reparadora, entonces* $p'_{|w_1}$ *está reparada.*

En otras palabras, la Propoción 5.1.2 establece que la reparación un error de corrección $e_1 \equiv (\mathtt{p}, w_1, \mathtt{l}_1, \sigma_1, C_1)$ permite arreglar automáticamente cualquier error que esté incluido en el término $\mathtt{p}_{|w_1}$. Pero, ¿qué pasa si los errores no son comparables w.r.t. $\preceq$?, o ¿qué sucede si se decide reparar un error que no sea el menor en el orden?. ¿Es aún posible reparar más de un error al mismo tiempo?. En lo siguiente, profundizaremos el análisis sobre la relación entre errores de corrección para responder a estas preguntas. Comenzaremos dando una función auxiliar.

Definición 5.1.3 (overlap) *Sean* $e_1 \equiv (\mathtt{p}, w_1, \mathtt{l}_1, \sigma_1, C_1)$ *y* $e_2 \equiv (\mathtt{p}, w_2, \mathtt{l}_2, \sigma_2, C_2)$ *dos errores de corrección en* $\mathbb{E}_N(\mathtt{p})$*. Decimos que* e_2 *solapa (o se superpone) con* e_1 *en* w *(en símbolos,* $e_2 \between_w e_1$*), sii*
(i) $e_1 \preceq e_2$*, y*
(ii) existe $w \equiv min(Emb_{\mathtt{l}_1}(\mathtt{p}_{|w_1}) \cap Emb_{\mathtt{l}_2}(\mathtt{p}_{|w_2}))$*, donde* $min(X) = w$ *s.t.* $w \leq w_i$ *para todo* $w_i \in X$*.*
Cuando la posición w *no es relevante, simplemente escribiremos* e_2 *solapa con* e_1 *o* $e_2 \between e_1$*. Por* $e_2 \not\between e_1$ *denotamos que* e_2 *no solapa con* e_1*.*

Dado dos errores de corrección e_1 y e_2 de una página Web $\mathtt{p}$, se distinguen tres posibles escenarios:

1. e_1 y e_2 son no comparables w.r.t. $\preceq$ (ver Figura 5.1(a));
2. $e_1 \preceq e_2$ y e_2 no solapa con e_1 (ver Figura 5.1(b));
3. e_2 solapa con e_1 (ver Figura 5.1(c)).

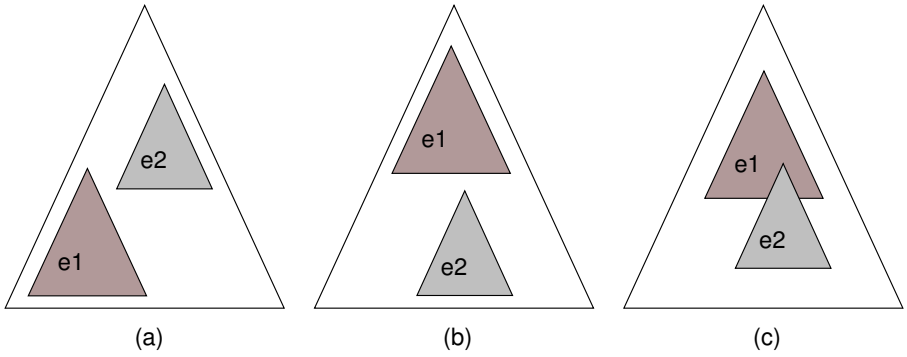

Figura 5.1: Distintos tipos de dependencias entre errores.

En el escenario 1, e_1 y e_2 son completamente independientes, y la reparación de uno no afecta a la corrección del otro. Este hecho, junto a la Proposición 5.1.2, permite una optimización obvia del marco de reparación, la cual esta formalizada en $\mathcal{M}$*-strategy* descrita más adelante en 5.1.9.

Considerando el escenario 2 y la Proposición 5.1.2, reparando e_1 se arregla automáticamente e_2. Pero al contrario no es cierto, puesto que reparar e_2 no ayuda a reparar e_1. Esto es establecido en la siguiente proposición.

Proposición 5.1.4 *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Sean dos errores de corrección* $e_1 \equiv (\mathtt{p}, w_1, \mathtt{l}_1, \sigma_1, C_1)$ *y* $e_2 \equiv (\mathtt{p}, w_2, \mathtt{l}_2, \sigma_2, C_2)$ *en* $\mathbb{E}_N(\mathtt{p})$*, tal que* $e_1 \preceq e_2$ *y* $e_2 \not\preceq e_1$*. Si* $\mathtt{p}' \equiv action(\mathtt{p}, w_2, t)$*,* $action \in \{\mathbf{delete}, \mathbf{change}\}$*, entonces*
(i) $p'_{|w_2}$ *es reparado,*
(ii) $(p', w_1, \mathtt{l}_1, \sigma', C_1) \in \mathbb{E}_N(\mathtt{p}')$ *para alguna substitución* σ'*.*

En el escenario 3, e_1 y e_2 están "conectados", es decir, $e_1 \preceq e_2$ y e_2 solape con e_1. Se podría pensar que el hecho de que e_2 esté parcialmente "contenido" en e_1 llevaría que la reparación de e_2 también repararía e_1. Esto no es siempre así. Considere el siguiente ejemplo.

Ejemplo 5.1.5 *Considere la página Web* $\mathtt{p} \equiv f(g(a), h(b))$ *y los siguientes errores de corrección* $e_1 \equiv (\mathtt{p}, \Lambda, f(h(X)), \{X/b\}, \emptyset)$*,* $e_2 \equiv (\mathtt{p}, 2, h(X), \{X/b\}, \emptyset)$*. Así,* $e_1 \preceq e_2$ *y* e_2 *solapa con* e_1*. Podríamos reparar* e_2 *cambiando la instancia* $h(b)$ *por* $h(a)$ *y, como se puede observar, esto no repararía automáticamente* e_1*, mientras que eliminado* $h(b)$ *o reemplazando* $h(b)$ *por el término* $l(c)$ *repararíamos* e_2 *y* e_1 *con una única acción.*

El Ejemplo 5.1.5 ilustra algunas condiciones necesarias para reparar automáticamente e_1 por medio de la reparación de e_2. En la siguiente proposición de detalla lo necesario para este propósito.

Proposición 5.1.6 *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Sean* $e_1 \equiv (\mathtt{p}, w_1, \mathtt{l}_1, \sigma_1, C_1)$ *y* $e_2 \equiv (\mathtt{p}, w_2, \mathtt{l}_2, \sigma_2, C_2)$ *dos errores de corrección en* $\mathbb{E}_N(\mathtt{p})$ *tal que* $e_1 \preceq e_2$ *y* e_2 *solapa con* e_1 *en* w*. Entonces tenemos que:*

- *Si* $\mathtt{p}' \equiv \mathbf{delete}(\mathtt{p}, w_2, t)$*, entonces*
  (i) $p'_{|w_2}$ *está reparado,*
  (ii) $(p', w_1, \mathtt{l}_1, \sigma') \notin \mathbb{E}_N(\mathtt{p}')$*, con la substitución* σ'*;*

- *Si* $\mathtt{p}' \equiv \mathbf{change}(\mathtt{p}, w_2, t)$ *y* $\mathtt{l}_{1|w} \not\trianglelefteq t$*, entonces*
  (i) $p'_{|w_2}$ *está reparado,*
  (ii) $(p', w_1, \mathtt{l}_1, \sigma') \notin \mathbb{E}_N(\mathtt{p}')$*, con la sustitución* σ'*;*

- *Si* $\mathtt{p}' \equiv \mathbf{change}(\mathtt{p}, w_2, t)$*,* $\mathtt{l}_{1|w}\sigma' \trianglelefteq t$ *para alguna sustitución* σ'*, y no se cumple* $C_1(\sigma_1/\sigma')$*, entonces*
  (i) $p'_{|w_2}$ *está reparado,*
  (ii) $(p', w_1, \mathtt{l}_1, \sigma_1/\sigma', C_1) \notin \mathbb{E}_N(\mathtt{p}')$*.*

5.1.2. Estrategias de corrección

Como vimos en el Capítulo 4, un error de corrección e en una página Web $\mathtt{p}$ puede ser reparado por la ejecución de una acción reparadora a. Denotaremos (e, a) al par que contiene una acción reparadora a que repara e. Con la notación $p' = a(p)$ denotamos que la ejecución de la acción reparadora a sobre la página Web p retornará la página Web p'

Definición 5.1.7 (estrategia de corrección) *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web, y* $\mathbb{E}(\mathtt{p}) = \{e_1, \ldots, e_n\}$ *un error de corrección de p. Una* estrategia de corrección *para* $\mathtt{p}$ *es una secuencia*

$$\langle (e_1, a_1), \ldots (e_2, a_n) \rangle,$$

donde $a_1, \ldots a_n$ *son acciones reparadoras tales que*

1. $\mathtt{p}_0 = \mathtt{p}$;

2. $\mathtt{p}_i = a_i(\mathtt{p}_{i-1})$, $0 < i \leq n$.

y $\mathtt{p}_n$ *está reparada.*

En otras palabras, dada un página Web con fallas $\mathtt{p}$, una estrategia de corrección para $\mathtt{p}$ permite reparar todos los errores en $\mathtt{p}$ por la ejecución de todas las acciones reparadoras que pertenecen a la estrategia.

Como vimos en la sección previa, reparar un error de corrección puede automáticamente reparar otros errores. Este hecho sugiere que una estrategia de corrección no necesariamente necesite contar con el par (e, a) para cada error e que aparece en la página Web. Basándonos en este resultado, a continuación se describen dos posibles estrategias de corrección: la primer estrategia minimiza el número de acciones necesarias para reparar la página Web; la segunda reduce la cantidad de información a ser eliminada/cambiada para corregir la página Web. En ambos casos, se asume que, para cada $e \in \mathbb{E}_N(\mathtt{p})$, se tiene un par *error/acción* (e, a). Y llamaremos $\mathbb{EA}(\mathtt{p})$ al conjunto que contiene estos pares.

A. Estrategia: Minimizar el número de acciones

En primer lugar, daremos un orden parcial sobre $\mathbb{EA}(\mathtt{p})$, el cual está inducido por el orden $\preceq$ sobre los errores de corrección.

Definición 5.1.8 *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Dados* (e_1, a_1), $(e_2, a_2) \in \mathbb{EA}(\mathtt{p})$,

$$(e_1, a_1) \sqsubseteq_T (e_2, a_2) \text{ sii } e_1 \preceq e_2.$$

Decimos que $(e, a) \in \mathbb{EA}(\mathtt{p})$ *es* minimal *w.r.t.* $\sqsubseteq_T$ *sii no existe* $(e', a') \in \mathbb{EA}(\mathtt{p})$ *s.t.* $(e', a') \sqsubseteq_T (e, a)$.

Ahora, observemos los siguientes hechos.

Hecho 1: Por la proposición 5.1.2, tenemos que, para cualquier $(e, a), (e', a') \in \mathbb{EA}(\mathtt{p})$ tal que $(e, a) \sqsubseteq_T (e', a')$, la ejecución de la acción reparadora a reparará a e y e'. De esta manera, reparar cualquier error de corrección e que corresponda a el minimal $(e, a) \in \mathbb{EA}(\mathtt{p})$ w.r.t. $\sqsubseteq_T$ reparará cualquier error e' que se mayor que e sin necesidad de ejecutar otra acción.

Hecho 2: Dado (e_1, a_1), $(e_2, a_2) \in \mathbb{EA}(\mathtt{p})$ ambos minimales w.r.t. $\sqsubseteq_T$, entonces e_1 y e_2 no son comparables w.r.t. $\preceq$.

De estos hechos, está claro que, para reparar toda la página Web, será suficiente reparar los errores que corresponden a los pares minimales.

Definición 5.1.9 ($\mathcal{M}$-*strategy*) *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web, y* $\mathbb{E}(\mathtt{p})$ *un conjunto de errores de corrección en p. Una estrategia* minimal *(o* $\mathcal{M}$-strategy*) para* p *es una secuencia* $\langle (e_1, a_1), \ldots (e_m, a_m) \rangle$,$(e_i, a_i) \in \mathbb{EA}(\mathtt{p})$, $i = 1, \ldots m$, *s.t. cada* (e_i, a_i) *es minimal w.r.t.* $\sqsubseteq_T$.

Proposición 5.1.10 *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Entonces la* $\mathcal{M}$-strategy *para* p *es una estrategia de corrección para* p.

Ejemplo 5.1.11 *Considere la página Web*

$$\mathtt{p} \equiv f(g(10), h(d), 20)$$

junto con el siguiente par de error/accción:

$$\langle(\mathtt{p}, \Lambda, f(g(X), 20), \{X/10\}, \{X < 20\}), \mathbf{change}(p, \Lambda, f(g(20), 10))\rangle$$

y

$$\langle((\mathtt{p}, 2, h(Y), \{Y/d\}, \emptyset), \mathbf{delete}(\mathtt{p}, 2, h(d))\rangle$$

En este caso, la estrategia $\mathcal{M}$-strategy *corresponde a la secuencia unaria*

$$\langle(\mathtt{p}, \Lambda, f(g(X), 20), \{X/10\}, \{X < 20\}), \mathbf{change}(p, \Lambda, f(g(20), 10))\rangle$$

La estrategia de la Definición 5.1.9 minimiza el número de acciones necesarias a ser ejecutadas para reparar la página Web como lo define la siguiente proposición.

Proposición 5.1.12 *Sean* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web y* $\mathcal{T}$ *una* $\mathcal{M}$*-strategy para* p*. Entonces, para cada estrategia de corrección* $\mathcal{S}$ *para* p*,* $length(\mathcal{T}) \leq length(\mathcal{S})$*, donde* $length(\cdot)$ *computa el número de pares error/acción de una estrategia de corrección.*

B. Estrategia: Minimizar la cantidad de información a cambiar/eliminar

En general, la $\mathcal{M}$*-strategy* fuerza al usuario a modificar/introducir una gran cantidad de información en una página Web, cuando en algunos casos es necesario hacer un pequeño cambio para repararla. Considere el siguiente ejemplo.

Ejemplo 5.1.13 *Sea la página Web*

$$\mathtt{p} \equiv f(g(a), k(m(c)), h(a))$$

y el conjunto

$$\mathbb{E}_N(\mathtt{p}) = \{(\mathtt{p}, \Lambda, f(g(X), h(Y)), \{X/a, Y/a\}, \{X{=}Y\})\}, (\mathtt{p}, 1, g(a), \varepsilon, \emptyset)\}.$$

De los dos errores, la $\mathcal{M}$-strategy *repararía el que se encuentra en la raíz de la página (por ser este el minimal). Si lo que el usuario quiere es tener una variante de* p*, debería introducir una gran cantidad de información. Por ejemplo* $\mathbf{change}(\mathtt{p}, \Lambda, f(g(b), k(m(c)), h(a)))$*, el usuario debería re-introducir casi la página Web completa, solamente para un pequeño cambio en la posición* 1,1*. Ahora*

bien, si nosotros reparáramos el segundo error $(\mathtt{p}, 1, g(a), \varepsilon, \emptyset)$ *por medio de la acción* **change**$(\mathtt{p}, 1, g(b))$*, el usuario debería introducir una pequeña cantidad de información para reparar ambos errores.*

La idea de la estrategia que vamos a definir es reparar los errores de corrección que estén en los niveles más bajos como se posible, y propagar la reparación hacia la raíz.

Obviamente, dados dos errores de corrección e y e' tal que $e' \preceq e$, corregir e no garantiza la reparación automática de e'. Esto está establecido en la Proposición 5.1.4 que dice, siempre que un error e' no solape a un error e, entonces no es posible extender la corrección de e sobre e'. Pero, por otro lado, bajo ciertas condiciones, la superposición de errores permite inferir una reparación sobre e' reparando e (ver Proposición 5.1.6).

De esta manera, la estrategia trabaja como sigue. En primer lugar, dada una página Web $\mathtt{p}$, se define una partición sobre $\mathbb{E}_N(\mathtt{p})$ de esta manera:

- $\mathbb{N}\mathrm{OVL}(\mathtt{p}) = \{e \in \mathbb{E}_N(\mathtt{p}) \mid \nexists\, e', e' \between e\}$
- $\mathcal{O}\mathrm{VL}(\mathtt{p}) = \mathbb{E}_N(\mathtt{p}) \setminus \mathbb{N}\mathrm{OVL}(\mathtt{p})$.

Claramente, $\mathbb{E}_N(\mathtt{p}) = \mathbb{N}\mathrm{OVL}(\mathtt{p}) \cup \mathcal{O}\mathrm{VL}(\mathtt{p})$. Llamaremos a los errores en $\mathbb{N}\mathrm{OVL}(\mathtt{p})$ (resp., en $\mathcal{O}\mathrm{VL}(\mathtt{p})$) errores *non-overlapping* (resp., errores *overlapping*).

Note que un error *non-overlapping* e no puede ser automáticamente reparado por la ejecución de una acción reparadora sobre un error e' tal que $e \preceq e'$, debido a que no se puede propagar el efecto hacia arriba. Sin embargo, este es el caso de los errores *overlapping* que pueden ser implícitamente afectados por otras reparaciones. Consideremos los siguientes hechos.

Hecho 1. Dado un error *overlapping* e, debe existir un error *non-overlapping* e' s.t. $e \preceq e'$.

Hecho 2. Sea $e, e_0, e_1, \ldots e_n$, $n \geq 0$, un error de corrección. Si e es un error *overlapping* tal que e_0 solapa con e y $e \preceq e_n \preceq e_{n-1}, \ldots \preceq e_0$, entonces e_i solapa con e, $i = 1, \ldots n$.

Juntando estos hechos con la Proposición 5.1.2 tenemos que:
(i) todos los errores mayores w.r.t. $\preceq$ que los errores considerados *non-overlapping* serán reparados por lo establecido en la Proposición 5.1.2; (ii) para cada error *overlapping* e, existe siempre un $e' \in \mathbb{N}\text{OVL}(\mathtt{p})$ que solape con e; por lo tanto, reparando e' también se repara e siempre que se cumpla la siguiente propiedad de seguridad (*safety*).

Definición 5.1.14 (safety) *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Sean* $e \equiv (\mathtt{p}, w, \mathtt{l}, \sigma, C) \in \mathbb{N}OVL(\mathtt{p})$*, y* $(e, \mathbf{change}(\mathtt{p}, w, t)) \in \mathbb{EA}(\mathtt{p})$*. Entonces, la propiedad de* safety *para* $(e, \mathbf{change}(\mathtt{p}, w, t)) \in \mathbb{EA}(\mathtt{p})$ *establece que, para cada* $e' \equiv (\mathtt{p}, w', \mathtt{l}', \sigma', C') \in \mathcal{O}VL(\mathtt{p})$ *s.t.* $e' \preceq e$*, una de las siguientes condiciones deben cumplirse:*

- $\mathtt{l}'_{|w} \not\trianglelefteq t$*, o*
- $\mathtt{l}'_{|w}\sigma' \trianglelefteq t$ *para alguna substitución* σ'*, y no se cumple* $C'(\sigma/\sigma')$*.*

Note que la propiedad de *safety* viene directamente de la Proposición 5.1.6, la cual garantiza la propagación automática de la reparación. Por otra parte, observe que esta propiedad sólo afecta a las acciones de tipo *change*, debido a que las acciones de borrado siempre permiten la propagación de la corrección.

Consideremos el siguiente orden parcial sobre los elementos en $\mathbb{N}\text{OVL}(\mathtt{p})$.

Definición 5.1.15 *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Dado* (e_1, a_1)*,* $(e_2, a_2) \in \mathbb{N}OVL(\mathtt{p})$*,*

$$(e_1, a_1) \sqsubseteq_B (e_2, a_2) \text{ sii } e_1 \preceq e_2.$$

Decimos que $(e, a) \in \mathbb{N}OVL(\mathtt{p})$ *es* minimal *w.r.t.* $\sqsubseteq_B$ *sii no existe* $(e', a') \in \mathbb{N}OVL(\mathtt{p})$ *s.t.* $(e', a') \sqsubseteq_B (e, a)$*.*

Ahora, estamos listos para proveer la estrategia minimal *non-overlapping*.

Definición 5.1.16 ($\mathcal{MNO}$-strategy) *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web, y* $\mathbb{N}OVL(\mathtt{p})$ *un conjunto de errores de corrección* non-overlapping *de p. Una estrategia* minimal non-overlapping *(o $\mathcal{MNO}$-strategy) para* p *es una secuencia* $\langle (e_1, a_1), \ldots (e_m, a_m) \rangle$*,* $(e_i, a_i) \in \mathbb{EA}(\mathtt{p})$*,* $i = 1, \ldots m$*, tal que*

- *cada* (e_i, a_i) *es minimal w.r.t.* $\sqsubseteq_B$ *y* $e_i \in \mathbb{N}OVL(\mathtt{p})$*;*
- *si* a_i *es una acción de tipo* change*, entonces se debe cumplir la propiedad de* safety *para* (e_i, a_i)*.*

Proposición 5.1.17 *Sea* $\mathtt{p} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ *una página Web. Entonces la* $\mathcal{MNO}$-strategy *para* p *es una estrategia de corrección para* p*.*

En la Figura 5.2, se muestra cómo trabaja la estrategia $\mathcal{MNO}$. Por una cuestión de simplicidad, cada nodo de la página está etiquetado como sigue:

- *ok*, si esta posición no hay tiene un error.
- *ov*, si esta posición tiene un error de tipo *overlapping*.
- *no*, si esta posición tiene un error de tipo *non-overlapping*.

La página Web contiene 9 errores, pero para reparar la página completa es necesario reparar sólo 3 errores. Precisamente, estos errores corresponden a los errores *non-overlapping* minimales de la página.

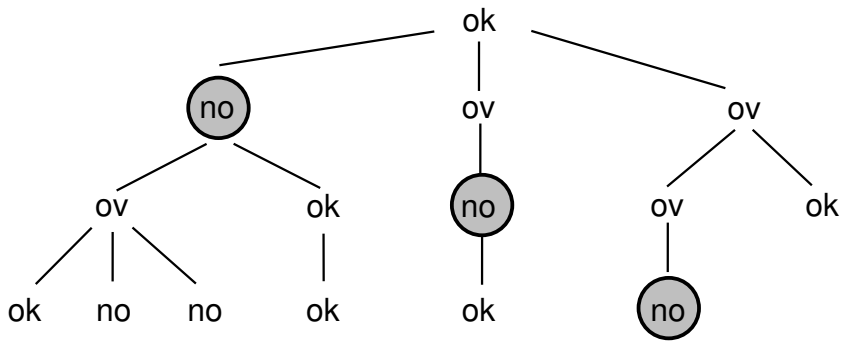

Figura 5.2: La $\mathcal{MNO}$-strategy

Ejemplo 5.1.18 *Considere la página Web*

$$\mathtt{p} \equiv f(g_1(h_1(a_1, a_2), h_2(b_1, b_2)), g_2(h_3(c_1), h_4))$$

y

$$\begin{aligned}\mathbb{E}_N(\mathtt{p}) = \{&(\mathtt{p}, 1, g_1, \varepsilon, \emptyset),\\ &(\mathtt{p}, 1{,}1, h_1(a_2), \varepsilon, \emptyset), (\mathtt{p}, 1{,}1{,}2, a_2, \varepsilon, \emptyset),\\ &(\mathtt{p}, 2, g_2(h_3(X)), \{X/c_1\}, \emptyset), (\mathtt{p}, 2{,}1, h_3(c_1), \varepsilon, \emptyset),\\ &(\mathtt{p}, 2{,}1{,}1, c_1, \varepsilon, \emptyset)\}\end{aligned}$$

por lo tanto,

$$\begin{aligned}\mathbb{N}OVL(\mathtt{p}) = \{&(\mathtt{p}, 1, g_1, \varepsilon, \emptyset), (\mathtt{p}, 1{,}1{,}2, a_2, \varepsilon, \emptyset),\\ &(\mathtt{p}, 2{,}1{,}1, c_1, \varepsilon, \emptyset)\}\end{aligned}$$

$$\begin{aligned}\mathcal{O}VL(\mathtt{p}) = \{&(\mathtt{p}, 1{,}1, h_1(a_2), \varepsilon, \emptyset), (\mathtt{p}, 2, g_2(h_3(X)), \{X/c_1\}, \emptyset),\\ &(\mathtt{p}, 2{,}1, h_3(c_1), \varepsilon, \emptyset)\}\end{aligned}$$

La estrategia $\mathcal{MNO}$ corrije solamente los errores que non-overlapping. *De esta manera, una estrategia $\mathcal{MNO}$ para* p *podría ser:*

$$\begin{aligned}\langle\!\langle &(\mathtt{p}, 1, g_1, \varepsilon, \emptyset), \mathbf{delete}(\mathtt{p}, 1, g_1(h_1(a_1, a_2), h_2(b_1, b_2)),\\ &((\mathtt{p}, 2{,}1{,}1, c_1, \varepsilon, \emptyset), \mathbf{change}(\mathtt{p}, 2{,}1{,}1, c_4))\rangle\!\rangle.\end{aligned}$$

Note que, se cumple la propiedad de seguridad para

$$((\mathtt{p}, 2{,}1{,}1, c_1, \varepsilon, \emptyset), \mathbf{change}(\mathtt{p}, 2{,}1{,}1, c_4))$$

. Con la ejecución de la estrategia de corrección se obtiene la página Web $f(g_2(h_3(c_4), h_4))$.

Finalmente, observemos que para obtener una páginas correcta, se necesito reparar sólo dos de los seis errores detectados originalmente.

5.2. 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 5.2.1 (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 σ 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 σ' 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 5.2.1, $s_0 \rightharpoonup^*_{I_M} \rightharpoonup s_{n-1} \rightharpoonup_{I_M} s_n$ 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 5.2.2 *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 dueño 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* $\sigma = \{X/\text{'Daniel Romero'}\}$.

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

5.2.1. 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 5.2.3 *(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 5.2.4 *(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 5.2.3 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 5.2.4, 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 5.2.5 *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 5.2.3 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 5.2.5 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 5.2.5 y el Algoritmo 1, permiten una optimización obvia del marco de reparación, que formalizamos en las estrategias de reparación presentadas a continuación.

5.2.2. Estrategias de reparación

Cómo explicamos en la Sección 5.2.1, 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 5.2.6 (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 5.2.1 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 5.2.5, 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 5.2.7 (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*

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

La Definición 5.2.7 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 5.2.8 *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 5.2.5, 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

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

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

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

Definición 5.2.9 (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*

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

La estrategia de la Definición 5.2.9 ejecuta el procedimiento *repairByInsert* en cada conjunto de la partición de $\mathbb{E}_M$.

Proposición 5.2.10 *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 5.2.11 *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 5.2.3 y 5.2.4 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 5.2.2 y 5.2.2 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

Extensiones y aplicaciones

En este capítulo se presentan dos trabajos que son derivados o extensiones de los trabajos de Verificación de sitios Web, descrito en el Capítulo 3, y la metodología de reparación, formulada en el Capítulo 4.

En primer lugar, se presenta una extensión de la metodología de verificación que permite analizar información no demandada *(undemanded)* dentro de un sitio Web y, de esta manera, detectar una tercera categoría de errores sobre el sitio Web, como así también incorporar nuevas reglas a la especificación [Alpuente et al., 2007g]. Por último, como consecuencia de la habilidad adquirida en las técnicas de *reescritura parcial* y *homeomorphic embedding*, se presenta un lenguaje de filtrado de documentos XML [Ballis and Romero, 2007b].

6.1. Análisis de información no demandada

En la metodología de verificación presentada en el Capítulo 3 se identifican dos categorías de errores: información incorrecta e información faltante dentro de sitio Web.

Cómo una extensión a esta propuesta, en [Alpuente et al., 2007g] se presenta una tercer categoría de errores que considera la información *undemanded* con respecto a una especificación Web.

En esta última categoría se intenta detectar posibles errores que puedan ser ocasionados por información no solicitada por el usuario. Veamos un ejemplo.

Ejemplo 6.1.1 *Considere W el sitio Web de la Figura 3.2, y E la especificación de la Figura 3.3.*

La regla de completitud r_5 de E establece que, para hacer un comentario en el blog, es necesario ser miembro del blog. Esta regla se cumple en W.

Observado las páginas de W encontramos que el miembro $PeterPan$ no tiene comentarios realizados.

De esto se desprende la siguiente interrogante: ¿es correcto o no, que algunos de los miembros del blog no tengan comentarios realizados?.

Teniendo presente el interrogante planteado en el Ejemplo 6.1.1, en [Alpuente et al., 2007g], se estudia y analiza este tipo de situaciones y se presenta un algoritmo capaz de: detectar posibles errores de *undemandedness*, interactuar con el usuario para resolverlos, e incorporar nuevas reglas de completitud a la especificación.

A continuación se describen los aspectos más importantes de la propuesta.

Una primera opción para definir *información undemanded* sería decir lo siguiente:

Información undemanded *es la información excedente conocida, donde "conocida" es que pertenezca a la especificación.*

Esto, en el contexto de la verificación de sitios Web, sería cualquier cosa que esté o no incluida en la especificación, lo que significaría chequear cada posible pieza de información incluso si no esté especificada. Pero esto no tiene sentido debido a que la semántica del lenguaje de especificación no ha sido diseñada para incluir todos los elementos que pueden estar presentes, sino definir algunas relaciones entre ellos.

La idea, para tratar con la información *undemanded*, es verificar los datos que son mencionados en la especificación.

Esto permite focalizar la información que el usuario consideró interesante en un comienzo (cuando definió la especificación), como así también delimitar el espacio de trabajo.

En este sentido, la definición de información *undemanded* es:

Definición 6.1.2 (información *undemanded*) *Se define como* información undemanded *al dato que está definido por la especificación y se usa en una estrucura diferente a la especificada.*

En el contexto de la verificación de sitios Web presentada en el Capítulo 3, la información requerida está especificada en las reglas de completitud[1]. Por esta razón, es éste conjunto de reglas las que se tienen presente para analizar la información *undemanded*.

La idea en el análisis de información *undemanded* es procesar las reglas de completitud $l \rightharpoonup r$ de la especificación Web de forma que, si en el sitio Web está presente una expresión simulada por la parte derecha r de la regla, sin que esté presente la parte izquierda l, esta expresión será no requerida a menos que exista una regla $r \rightharpoonup l$ incluida en la especificación.

De esta manera es posible obtener nuevas reglas de completitud derivadas de la reglas de completitud iniciales.

Con las reglas derivadas de este proceso, es posible detectar errores de información *undemanded* con respecto a su especificación usando la ejecución del motor Verdi-M. Pero no toda la información *undemanded* detectada será un error, por lo que es necesario consultar al usuario para clasificar los datos encontrados por el algoritmo.

De esta clasificación, el algoritmo puede decidir si las reglas serán grabadas para una próxima iteración y agregadas a la especificación original, o simplemente descartarlas.

Cuando una regla se agrega a la especificación, el algoritmo verifica que se continúe cumpliendo la propiedad de *boundedness* definida en [Ballis, 2005] que garantiza la terminación del proceso de verificación. De esta manera, si una regla no puede ser agregada, se retorna el conjunto de errores detectados por ella para que el usuario pueda solucionarlos.

El algoritmo es capaz de aprender de la información entrada por el usuario y de esta manera refinar la especificación.

Esto se realiza por medio de una operación que marca las reglas de completitud.

A continuación describiremos los seis pasos que componen el algoritmo: preparación, filtrado, inversión, chequeo, búsqueda y aprendizaje.

- Preparación: Debido a múltiples iteraciones posibles, algunas reglas son usadas en el algoritmo. Por ejemplo, si una regla se invierte, sería invertida

[1]En las reglas de corrección se define cuál es la información que no se permite en el sitio.

mas de una vez.

- Filtrado: No son consideradas interesantes las reglas $l \rightharpoonup r$ en donde $r \trianglelefteq l$, porque luego de invertir la regla, $r \rightharpoonup l$ siempre sería válida. De esta manera también se garantiza mantener la propiedad de *boundedness* de la especificación.

- Inversión: Las reglas obtenidas del paso de *filtrado* son invertidas usando el operador *reverting*.

- Chequeo: Aquí se verifica si la especificación original, junto con las reglas invertidas, permiten alcanzar todas las páginas del sitio. De este paso se puede concluir que una página no alcanzada podría ser información *undemanded* o que la especificación es incompleta.

- Búsqueda: Las reglas invertidas son dadas como la especificación de entrada para Verdi-M. De esta manera, los errores de completitud detectados serán los posibles errores de undemandedness.

- Aprendizaje: Los posibles errores de undemandedness detectados, son dados al usuario para su validación.

  El usuario tiene diferentes posibilidades para hacer: confirmar el error como error de *undemandedness* o decir que es información correcta.

  Cuando la información se considera correcta, resulta necesario agregarla (como una regla de completitud) a la especificación original para que no se vuelva a detectar como error.

  Para el caso en que sea correcta pero la información sea particular, se agrega la instancia de la regla detectada.

En conclusión, la ejecución del algoritmo de análisis de *undemandedness* agregará reglas de completitud a la especificación original. De esta manera es posible detectar errores de *undemandedness* que no fueron considerados por el usuario.

Este algoritmo fue implementado como una extensión del prototipo WebVerdi-M descrito en el Capítulo 8.

6.2. Filtrado de documentos XML

Haciendo reuso de la técnica de *reescritura parcial* (ver Sección 3.3), en [Ballis and Romero, 2007b] se presenta un lenguaje de filtrado para documentos XML. El lenguaje definido permite al usuario fácilmente seleccionar o eliminar información dentro de un documento XML. Se considera que el lenguaje definido posee las características necesarias que se requieren para hacer un filtrado, como así también una sintaxis simple acompañada de una clara e intuitiva semántica.

La notación utilizada y los conceptos preliminares fueron definidos previamente en la Sección 2. Por una cuestión de contextualizar los nombres utilizados, llamaremos documento XML a una página Web, y plantilla de docuemtno XML a una plantilla de página Web (ver la definición de estos conceptos en la Sección 3.1).

A continuación se describen las características generales del trabajo.

6.2.1. Lenguaje de filtrado

Basicamente, una regla de filtrado formaliza el patrón de información (también llamada criterio de filtrado) que será detectado dentro del documento XML `p`. La información, que es reconocida en `p`, es seleccionada (filtrado positivo) o eliminada (fitrado negativo), siempre que la condición de filtro sobre la información detectada sea satisfecha. El criterio de filtro se especifica con una plantilla de documento XML.

La condición de filtrado consiste en una secuencia de ecuaciones o expresiones. El detalle de las mismas se encuentra en la Sección previa 3.2.

En la Figura 6.1, se presenta la gramática completa del lenguaje de filtrado utilizando una notación BNF. Tres aspectos son los más importante a destacar de la gramática, *criterion*, *expression* y *condition*.

- *criterion*: Representa el patrón de información a ser detectado dentro del documento XML. Es especificaco usando una plantilla de documento XML.
- *expression*: Es el documento XML a ser examinado.
- *condition*: Es una secuencia de ecuaciones o expresiones regulares a ser cumplidas por los términos del documento. Es posible utilizar operaciones

```
(1)  <filterRule> ::= <filter> <criterion> in
                 <expression> [where <condition>]
                 [label]
(2)  <filter> ::= filter | filter*
(3)  <criterion> ::= <xmlDocumentTemplate>
(4)  <expression> ::= <xmlDocument> | (<filterRule>)
(5)  <condition> ::= <sequence of membership tests
                      and equations>
(6)  <label> ::= (P) | (N)
```

Figura 6.1: BNF de la gramática para el lenguaje de filtrado de XML

definidas por el usuario.

A continuación, se describe la semántica de cada una de las opciones del lenguaje, acompañada de un ejemplo práctico.

6.2.2. Filtrado positivo

Esencialmente, dado una regla de filtro *filter* t *in* p *where* C (P), la idea es computar el conjunto de todas las substituciones $\{\sigma_1, \ldots, \sigma_n\}$ tal que t sea embebida en p. De esta manera, se junta cada $t\sigma_i$, tal que se cumple la condición C por la σ_i, en un nuevo documento XML.

Ejemplo 6.2.1 *Dada la regla de filtro positivo*

filter *book(title(X),author(surname(Y)), code(W))*
in *book(title(El Alquimista), author(name(Paulo),*
surname(Coelho)),year(2002), abstract(blablabla), code(PC))
where *X in [:Text:]Alquimista[:Text:], W = first(Y) ++ first(Z)*
(P)

se obtiene el siguiente resultado

book(title(El Alquimista), author(surname(Coelho)), code(PC))

6.2.3. Filtrado negativo

Dada la regla de filtro negativa *filter t in p where C* (N), el proceso elimina cada instancia $t\sigma$ de t que esté embebida en p siempre que se cumple la condición asociada C.

Ejemplo 6.2.2 *Sea una regla de filtro negativa*

filter *book(code(X),name(Y))*
in *book(title(El Alquimista), author(surname(Coelho),*
name(Paulo)),year(2002), abstract(blablabla),
code(PC))
(N)

El resultado obtenido es el siguiente

book(title(El Alquimista), author(surname(Coelho)), abstract(blablabla),year(2002))

6.2.4. Composición de reglas de filtrado

El lenguaje de filtro definido permite al usuario especificar reglas de filtros complejas por medio de la combinación de reglas más simples (ver las reglas 1 y 4 de la gramática descrita en 6.1). Esto es llamado composición de reglas. Veamos un ejemplo de esta situación.

Ejemplo 6.2.3 *Sea la regla de filtro compuesta*

filter *name(X)* ***in***
(***filter*** *book(title(X),author(name(Y),*
surname(Z)),code(W))
in *book(title(El Alquimista),*
author(surname(Coelho),
name(Paulo)), year(2002),
abstract(blablabla),code(PC))
(P)
)
(N)

obtenemos

book(title(El Alquimista), author(surname(Coelho)), code(PC))

6.2.5. Filtrado ordenado

Cuando se conoce la estructura del documento XML, existe la posibilidad de utilizar la opción de *filtrado ordenado*. Esto exige que se respete el orden de los argumentos de los términos del criterio. Para ello se utiliza la palabra reservada *filter**.

Ejemplo 6.2.4 *Sea la regla de filtro*

filter* *book(code(X),title(Y))*
in *book(title(El Alquimista),*
author(surname(Coelho),
name(Paulo)),year(2002),
abstract(blablabla),
code(PC))

esto producirá como resultado un documento vacío, debido a que el orden entre code *y* title *no es respetado en el documento XML.*

7

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 7.1, introducimos el concepto de interpretación abstracta. En la Sección 7.2, describimos algunos trabajos relacionados y nuestra propuesta de transformación *source-to-source*. En la Sección 7.3, describimos como representar un sitio Web. La Sección 7.4 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 7.5 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 transformarse directamente al nivel abstracto.

7.1. 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 navegarlo por completo.. 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 interpretación abstracta es una aproximación para este problema.

Ilustraremos el significado 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 solo 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 "si 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 mienstras se mantiene la suficiente precisión para responder a questiones 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 los datos mismos. 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.

7.2. Interpretación abstracta aplicada al análisis de sitios Web

En la literatura, la interpretación abstracta ha sido escásamente 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 subyasente 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, este trabajo 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 rapidamente 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 la *performance* 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 paramétrico 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.

7.3. 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 7.3.1 (sucesor inmediato) *La relación de* sucesor inmediato *(*immediate successors*) para una página Web p se define por*

$$\rightarrow_p = \{(p, p') \subseteq \mathcal{P} \times \mathcal{P} \mid p' \text{ es directamente accesible desde } p\}$$

La Definición 7.3.1 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 7.3.2 (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 7.3.2 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.

7.4. 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

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

$drop\_duplicates(x : xs) =$

if $member(x, xs)$ then $drop\_duplicates(xs)$

else $x : drop\_duplicates(xs)$

Definición 7.4.1 (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 7.4.1 se ilustra en la Figura 7.1. Intuitivamente, primero se unen todos los argumentos con el mismo símbolo raíz ocurrido al mismo nivel i, luego, la compresión procede de manera recursiva al nivel $(i+1)$. La Figura 7.1(b) muestra la compresión del término de la Figura 7.1(a).

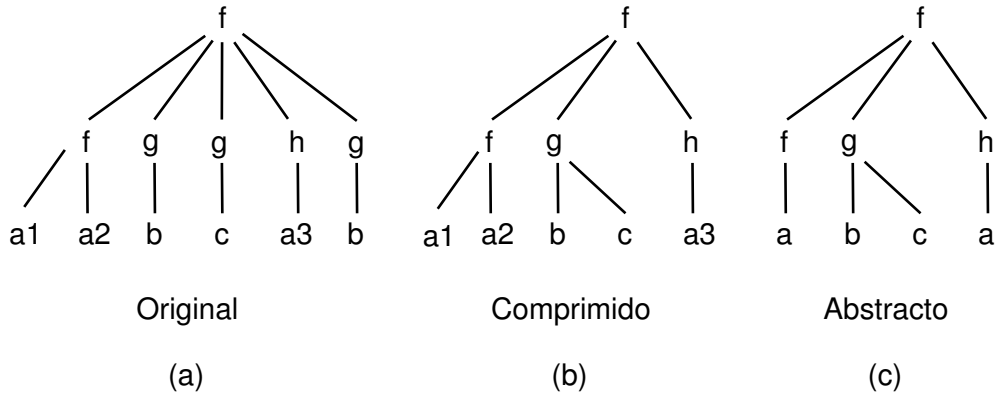

Figura 7.1: compresión y abstracción un término

7.5. Verificación de sitios Web abstractos

7.5.1. 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 7.5.1 (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^α) de un término t como: $t^\alpha = \alpha(t)$. Nuestro marco es paramétrico w.r.t. la función de abstracción α, la cual puede ser usada para ajustar la precisión de la aproximación.

Definición 7.5.2 (función de abstracción de términos α)

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

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 7.5.2, los elementos de $\mathcal{T}ext$ se abstraen tomando en cuenta la cadena de *tags* bajo la cual aparece una piesa 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 α, 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 7.5.3 *Considere nuevamente el término t en la Figura 7.1(b). Considerando $\alpha(w) = first(show\ w)$, donde $first(x : xs) = x$, la abstracción $\alpha(t)$ se muestra en la Figura 7.1(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 7.5.4 (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^α (resp. w_N^α) 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 $(I_N^\alpha, I_M^\alpha, R)$.

7.5.2. 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 ejemlo, considere $p_1, p_2 \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ s.t. $p_1 = professor(name(P.Almodovar))$ y $p_2 = professor(name(P.Cruz))$, considere también la función de abstracción α dada en el Ejemplo 7.5.3, entonces $\alpha(p_1) = \alpha(p_2) = professor(name(P))$.

Al mismo tiempo, la función de compresión de términos dada en la Sección 7.4 contribuye aún más a reducir el tamaño de cada página Web (ver Figura 7.1).

Por otro lado, para visitar o navegar un sitio Web (considerando la Definición 7.3.2), 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 7.5.5 (Sitio Web abstracto) *Sea W un sitio Web y p la página inicial de W. Entonces, la abstración de W se define por:*

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

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 7.5.5, obtener una representación finita abstracta de nuestro sitio Web.

7.5.3. Verificación de sitios Web abstractos

La función de absracción dada en la Definición 7.5.2 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^α 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^α w.r.t. $(I_N^\alpha, I_M^\alpha, R)$.

Definición 7.5.6 *Sea W un sitio Web y (I_N, I_M, R) una especificación Web, y sean W^α 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^α 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^α, entonces $\alpha(t_2) \preceq t'$. En símbolos tenemos

$$t_1 \rightharpoonup_r t_2 \wedge \alpha(t_1) \rightharpoonup_{r^\alpha} t' \Rightarrow \alpha(t_2) \preceq t' \tag{7.1}$$

Cuando esto pasa, decimos que la abstracción α (o mas precisamente la función $\alpha\_(c, f)$ que es un parámetro de la transformación de abstracción) es segura (*safe*) w.r.t. (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 7.5.7 *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^α *las correspondientes versiones abstractas de* (I_N, I_M, R) *y* W*. Si la abstracción* α *es segura w.r.t.* (I_N, I_M, R) *y* W *entonces cada error de corrección (resp., completitud) abstracto señala un error de corrección (resp., completitud) w.r.t.* (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 balance conveniente entre la eficiencia y la precisión del análisis para cada dominio específico.

8

Implementación y evaluación experimental

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

8.1. 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.

8.2. 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., 2006; 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 representación 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 8.1, 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

```
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 8.1: Ejemplo de codigo Maude

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 σ 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 proce-

so 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.

8.2.1. 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 σ 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

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

donde `sub` es un operador asociativo utilizado para registrar la sustitución σ 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 8.2). 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 σ como una secuencia de variables instanciadas X_i, $\texttt{i} = 1, \ldots, \texttt{n}$

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

Ejemplo 8.2.1 *Sean s_1 y s_2 dos plantillas de documentos XML definidos como sigue*

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

$$s_2 \equiv hpage(name(mario), surname(rossi), status(prof.), \\ teaching(course(logic1), course(logic2)), \\ hobbies(hobby(reading), hobby(gardening)))$$

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

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

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*

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

8.3. Prototipo WebVerdi-M

Junto con el avance de la línea de investigación, se desarrollo 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, se ha 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 estándars, con el objetivo de construir redes de aplicaciones colaborativas.

WebVerdi-M, al ser una arquitectura orientada a servicios, nos permite acceder al nucleo del motor de verificación Verdi-M como una entidad reusable. La implementación está disponible en `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* provee 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 8.2 ilustramos la arquitectura del sistema.

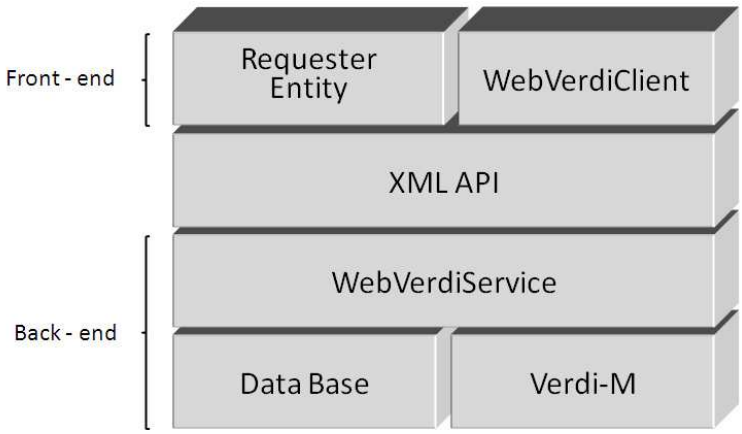

Figura 8.2: Componentes de WebVerdi-M

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 descriptas en el Sección 8.2, y son básicamente las propiedades de asociatividad y conmutatividad que permiten una simple y elegante implementación del algoritmo de simulación.

El servicio Web exporta seis operaciones que son accesibles a través mensajes estándares XML. Estas operaciones permiten: almacenar un sitio Web, remover

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

un sitio Web, recuperar un sitio Web, agregar 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., 2007f]) 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., 2007f].

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., 2007f]) 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 directorio, lo cual permite una rápida adaptación 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 álgebra de términos. En la herramienta se puede trabajar de manera indistinta en cualquiera de las tres vistas. En la Figura 8.3, se muestra una captura de pantalla del cliente Web WebVerdiClient. Respetando la API definida, un usuario podría utilizar cualquier otro cliente para trabajar.

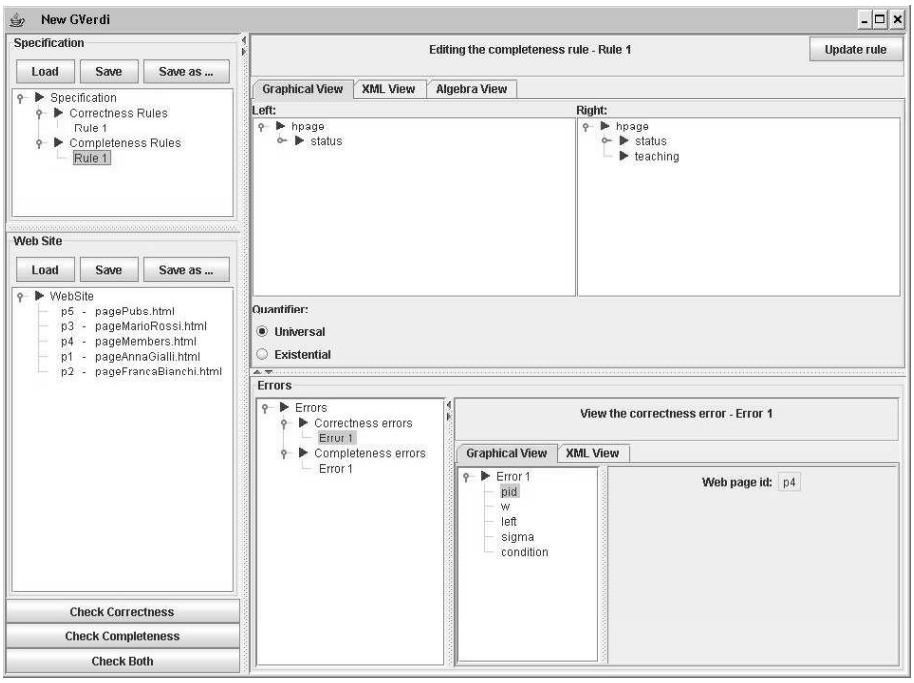

Figura 8.3: Captura de pantala del cliente Web WebVerdiClient

DB

En la interacción entre el servicio Web y la aplicación cliente se necesita transferir abundante información. Considere 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.

8.4. Evaluación experimental

Con el objetivo de evaluar la *performance* de Verdi-M en sistemas de tamaños reales, 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 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 8.1, 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 8.1 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 8.1: 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 eficiente y el crecimiento es lineal con respecto al número de nodos de los documentos. En la Tabla 8.1, 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 tomo 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 procesas 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 la *performance* obtenida en 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 solamente fue capaz de procesar pequeños repositores XML (no mayores a 1Mg) dentro de un tiempo razonable.

En la Capítulo 7 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.

9

Conclusiones y Trabajos futuros

9.1. 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], en donde se presenta un marco para la verificación de sitios Web. Dicho marco permite especificar restricciones de integridad para un sitio Web y entonces detectar con precisión las partes que no satisfacen la especificación. Una vez realizada la verificación resulta interesante poder corregir las anomalías encontradas.

Durante el primer periodo de esta tesis, continuamos esta línea de investigación definiendo una metodología de reparación semi-automática de los errores detectados [Alpuente et al., 2006b; Alpuente et al., 2006c]. Luego, mediante estrategias de reparación, se definieron estrategias que permitieron optimizar el proceso de reparación [Ballis and Romero, 2007b]. En este periodo también se

exploró el uso de la técnica de reescritura parcial aplicada al filtro de documentos XML [Ballis and Romero, 2007a].

En el segundo periodo, en [Alpuente et al., 2007d] completamos la metodología de reparación semi-automática de sitios Web 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.

También se investigó una tercera categoría de errores, que llamamos *undemandedness error*, en donde se considera la información innecesaria con respecto a una especificación Web [Alpuente et al., 2007g]. De esta manera, se obtuvo un algoritmo capaz de: detectar posibles errores de *undemandedness*, interactuar con el usuario para resolverlos, e incorporar nuevas reglas de completitud a la especificación original. Esta extensión está implementada en WebVerdi-M.

En [Alpuente et al., 2007c], 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 costo de ejecución de analizar sitios Web complejos.

Asimismo, en [Alpuente et al., 2007b; Alpuente et al., 2007e] realizamos un estudio de escalabilidad (en función del tiempo de ejecución y tamaño del sitio Web) de la metodología de verificación. Los resultados obtenidos son 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 (ver [Alpuente et al., 2006a]), excede el tiempo deseado y, en este caso, somos capaces de procesas 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.

Un implementación de nuestra metodología de se encuentra en el prototipo WebVerdi-M disponible en

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

9.2. Trabajos futuros

A continuación, se pretende describir posibles líneas de investigación que pudieran dar origen a trabajos futuros. Consideramos que los temas que serán abordados tienen entidad suficiente para dar lugar a la realización de una tesis doctoral.

9.2.1. Extensión del lenguaje de especificación

El análisis de información *undemanded* realizado en la Sección 6.1, despertó la posibilidad y conveniencia de extender el lenguaje de especificación. En primer lugar, queremos permitir condiciones lógicas más generales dentro de las reglas; esto nos permitirá elevar la capacidad expresiva de las mismas y dará al usuario la posibilidad de especificar condiciones mas precisas. Por otro lado, permitir definir reglas de completitud que no dependan de la presencia de alguna otra información relacionada.

9.2.2. Estrategias de reparación sobre las etiquetas de los nodos

En los Capítulos 3 y 5, vimos cómo es posible reparar los errores detectados sobre un sitio Web. Allí se definieron acciones reparadoras que son aplicadas sobre las páginas Web por medio de estrategias de reparación. En línea general, las estrategias presentadas trabajan sobre los sub-términos que se encuentran en los árboles (que representan las páginas Web), y las acciones reparadoras **change** y **delete** afectan a todos los subtérminos del término sobre el cual ellas se aplican. Observando el funcionamiento, una estrategia de reparación ingenuamente ideal podría ser aplicar **delete** a la raiz del árbol y, de esta forma, eliminar todos los errores de las páginas. Lamentablemente de esta manera, también se eliminaría toda la información de la misma.

Como opción al cambio o eliminación de todo el término, se podrían definir acciones reparadoras que trabajen afectando sólo la etiqueta del nodo donde se produce el error. De esta manera, se minimizaría la cantidad de información a ser manipulada por una acción y la estructura del sitio no se vería afectada.

Relacionado a éste problema, otro camino a considerar sería definir acciones que actuen sobre un subconjunto de los subtérminos.

9.2.3. Reducción del tamaño del sitio Web

Con el objetivo de acotar el grafo de recorrido de las páginas Web de un sitio Web, en el Capítulo 7 desarrollamos la idea de una "compresión horizontal" de documentos. Esto nos permitió detectar páginas Web que tienen estructuras similares pero contenidos diferentes.

El trabajo en esta línea se pretende ampliar estudiando también como aplicar una reducción vertical sobre las páginas Web. La idea es codificar los caminos de la estructura de la página en los nodos. Como una consecuencia de esta codificación, no sería directa la transformación *source-to-source* presentada y, por lo tanto, será necesario generalizar el *abstract embedding* dado.

9.2.4. Refinamiento automático de la función de abstracción

En el Capítulo 7, hemos presentado un marco abstracto de verificación que es paramétrico respecto de la función de abstracción. En ciertos dominios, el conocimiento del usuario sobre el sistema será suficiente para determinar una buena función de abstracción. Sin embargo, buscar una función de abstracción presenta en general la siguiente dicotomía. (i) La función es demasiado general como para obtener resultados reales; (ii) la función es demasiado precisa, y la verificación se hace inviable.

En este conexto, pretendemos investigar cómo maximizar la precisión del análisis por medio de refinamientos automáticos de la abstracción, preservando la eficiencia del proceso. Como marco de referencia, en [Clarke et al., 2000] se propone refinar la abstracción usando un algoritmo adaptativo que mejora la función de abstracción paso a paso guiado por *spurious conterexamples* (o fal-

sos contraejemplos). En nuestro framework, dichos falsos contraejemplos están asociados a los falsos errores detectados en el dominio abstracto.

9.2.5. Verificación de sitios Web dinámicos

Existen diferentes maneras en la cual se puede generar páginas Web dinámicas. Considere los siguientes ejemplos.

- Una página Web que implementa un calendario, siempre se puede hacer un click para ver el siguiente mes.
- La personalización de sitios Web. Si usted está navegando por el sitio Web `http://www.amazon.com` y selecciona sus libros favoritos, usted verá una lista de items junto a la lista de libros recomendados (generada de forma automática) con tópicos similares.
- Un gestor *online* de base de datos, en donde usted ingresa una consula SQL y en la siguiente página se muestran los resultados.
- En los sitios Web conocidos como bitácora (o *blogs*), en donde el número de páginas crece cuando el autor agregarse nuevas entradas, o las páginas existentes se modifican con nuevos comentarios de sus lectores.
- Sitios de noticias, ventas *online*, corporaciones, etc.

De la misma manera, se pueden imaginar una gran cantidad de situaciones y/o sitios Web en donde la generación de páginas dinámicas está presente.

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. Esto se debe a la secuencia infinita de páginas Web que se pueden ir generando de manera dinámica.

Con esta idea básica como fondo, nos proponemos investigar técnicas de verificación que sean independientes del lenguaje de *script* utillizado para la generación de las páginas. Es importante considerar las propiedades presentes en los sitios Web dinámicos (protocolos, sessiones, navegación, etc.) e integrarlas a nuestra metodología de verificación.

9.3. Cronograma de tesis doctoral

En esta sección se presenta un cronograma tentativo hacia la tesis doctoral. El plan de trabajo se organiza en tres bloques fundamentales que cubrirán algunos de los temas descriptos en la Sección 9.2. En líneas generales, primero plantearemos la extensión del lenguaje de especificación, en segundo lugar un análisis exautivo de la denominada Web dinámica y la incorporación de nuevos elementos en la metodología de verificación abstracta, por último, el estudio de técnicas para mejorar la precisión de la función de abstracción.

El plan de trabajo se estructura en las siguientes tareas:

- **Extensión del lenguaje de especificación:**
  - Revisión del marco formal.
  - Incorporación de condiciones lógicas a las reglas.
  - Extensión de la definición de las reglas de completitud.
  - Integración de conceptos en Verdi-M.
- **Metodología abstracta de verificación:**
  - Estudio exautivo de los componentes participantes en la generación de páginas Web dinámicas.
  - Estudio de la técnicas de interpretación abstracta.
  - Integración de los componentes dinámicos y las técnicas de interpretación abstracta Verdi-M.
- **Mejorar la precisión de la función de abstracción:**
  - Estudio de técnicas para refinar de forma automáticas la función de abstracción.
  - Integración de conceptos en Verdi.

Bibliografía

[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, An Abstract Generic Framework for Web Sites Verification, in *York Doctoral Symposium (YDS 2007)*, submited

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

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

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

[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

[Alpuente et al., 2006c] Alpuente, M., Ballis, D., Falaschi, M., and Romero, D.: 2006c, GVERDI-R A Tool for Repairing Faulty Web Sites, in *Proc. VI Jornadas sobre Programación y Lenguajes (PROLE'06). Sitges, Spain*, pp 221–230, CIMNE Barcelona

[Alpuente et al., 2007g] Alpuente, M., Romero, D., and Zanella, M.: 2007g, *A functional framework for analysis of undemandedness in websites*, Technical Report DSIC-II/14/07, DSIC, UPV

[Apt, 2003] Apt, K.: 2003, *Principles of Constraint Programming*, Cambridge University Press

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

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

[Ballis and Romero, 2007a] Ballis, D. and Romero, D.: 2007a, Filtering of XML documents, in *Proc of 2nd Int'l Workshop on Automated Specification and Verification of Web Systems (WWV'06). Paphos, Cyprus*, pp 19–26, IEEE Computer Society Press

[Ballis and Romero, 2007b] Ballis, D. and Romero, D.: 2007b, Fixing web sites using correction strategies, in *Proc of 2nd Int'l Workshop on Automated Specification and Verification of Web Systems (WWV'06). Paphos, Cyprus*, pp 11–19, IEEE Computer Society Press

[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

[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

[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/`

[Clarke et al., 2000] Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., and Veith, H.: 2000, Counterexample-guided abstraction refinement, in *Computer Aided Verification*, pp 154–169

[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

[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

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

[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)*

[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)*

[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

[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

[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

[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`

[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

[Projet CiME,] Projet CiME, *The* C*i*ME *Rewrite Tool*, Available at: `http://cime.lri.fr/`

[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

[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`

[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

Trabajos desarrollados en el marco de esta tesis

- **Internacionales**

  - M. Alpuente, D. Ballis, M. Falaschi y D. Romero
    A Semi-automatic Methodology for Reparing Faulty Web Sites.
    4th IEEE Int'l Conference on Software Engineering and Formal Methods (SEFM'06), Pune, India. IEEE Computer Society Press, pp 31–40, 2007.

  - D. Ballis y D. Romero
    Fixing Web Sites Using Correction Strategies.
    Specification and Verification of Web Systems (WWV'06). Paphos, Cyprus.
    IEEE Computer Society Press, pp 11–19, 2007.

  - D. Ballis y D. Romero
    Filtering of XML Documents.
    Specification and Verification of Web Systems (WWV'06). Paphos, Cyprus.
    IEEE Computer Society Press, pp 19–26, 2007.

  - 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.* York Doctoral Symposium (YDS 2007). October 26, 2007 – King's Manor, University of York, UK. **(Submited)**.

- **Nacionales**

  - 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, *to appear*.
  - M. Alpuente, D. Ballis, M. Falaschi y D. Romero GVERDI-R A Tool for Repairing Faulty Web Sites.
    Proc. VI Jornadas sobre Programación y Lenguajes (PROLE'06). Sitges, Spain, pp 221–230, CIMNE Barcelona
  - 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, *(submited)*.

- **Informes Técnicos (No publicados)**

  - M. Alpuente, D. Romero y M. Zanella
    A framework for analysis of undemanded data in Web sites.
    Technical Report DSIC-II/14/07. DSIC, UPV. Julio 10, 2007.