Estrategias de Reparación para Sitios Web Incompletos*

M. Alpuente[1], D. Ballis[2], M. Falaschi[3], P. Ojeda[1], and D. Romero[1]

[1] DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, Apdo. 22012, 46071 Valencia, Spain.
{alpuente, pojeda, dromero}@dsic.upv.es

[2] Dip. Matematica e Informatica
Via delle Scienze 206, 33100 Udine, Italy.
demis@dimi.uniud.it

[3] Dip. di Scienze Matematiche e Informatiche
Pian dei Mantellini 44, 53100 Siena, Italy.
moreno.falaschi@unisi.it

Abstract. El desarrollo y mantenimiento de sitios Web no es un tarea fácil. Garantizar que la información de un sitio Web es consistente resulta cada vez más dificil, por ello los administradores Web necesitan de mecanismos que ayuden a reparar estas inconsistencias. En este trabajo, completamos la metodología de reparación semiautomática de sitios Web erróneos presentada en un trabajo anterior con el tratamiento de los errores de completitud. Comenzamos con la categorización de dichos errores mediante dos nuevos órdenes que definimos sobre el conjunto de errores de completitud obtenidos en el proceso de verificación. Utilizando estos órdenes, definimos sendas estrategias capaces de minimizar la cantidad de acciones de reparación a realizarse para generar un sitio Web completo con respecto a su especificación.

Keywords: Verificación y reparación de sitios Web, reescritura, simulación, reescritura parcial, estrategias de reparación.

1 Introducción

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 sistemas y servicios web, y que permitan no sólo detectar automáticamente 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.

* This work has been partially supported by the EU (FEDER) and Spanish MEC TIN-2004-7943-C04-02 project, the Generalitat Valenciana under grant GV06/285, and Integrated Action Hispano-Alemana HA2006-0007. Daniel Romero is also supported by ALFA grant LERNet AML/19.0902/97/0666/II-0472-FA. Pedro Ojeda is also supported by the Generalitat Valenciana under FPI grant BFPI/2007/076.

En el marco de la reparación existen también diferentes líneas de investigación incipientes. En [17], se presenta un marco para reparar inconsistencias en documentos distribuidos, como un complemento a la herramienta de verificación xlinkit [11]. 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 de dichas reglas. Tampoco es posible detectar si dos expresiones que formulan un requerimiento para el sitio Web son incompatibles. Similarmente, en [18, 20] se presenta una extensión para CDET [19]. 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 se derivan las reparaciones directamente de este grafo. En este caso, las reglas temporales soportan interferencias y la compatibilidad de las reparaciones sí es tenida en cuenta. Desafortunadamente, esta compatibilidad resulta muy costosa de comprobar en reglas temporales. Ambas aproximaciones se basan en técnicas del campo de las bases de datos activas [8]. Otras investigaciones recientes en este campo se centran en la derivación de reglas activas que, automáticamente, lancen acciones reparadoras que conducen a un estado consistente después de cada actualización [16].

En trabajos previos [6, 2], presentamos 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: error de corrección (*correctness error*) y error de completitud (*completeness error*).

Tomando como base el marco de verificación de [2], en [3] hemos formulado también una metodología para la corrección del código erróneo. En ella, partiendo de una categorización de los diferentes tipos de error que pueden encontrarse en un sitio Web respecto a su especificación inicial, clasificamos las distintas clases de acciones de reparación que pueden ejecutarse para corregir cada error dado.

Cómo complemento a este trabajo, en [7] realizamos un análisis sistemático sobre la relación entre los errores de corrección detectados por nuestra metodología y, de esta manera, definimos dos estrategias, una para minimizar el número de acciones de reparación a ser ejecutadas y la otra para minimizar la información a ser cambiada o eliminada sobre el sitio Web. El resultado de la fase de corrección es un sitio Web correcto w.r.t. a su especificación aunque no completo.

Nuestra contribución. En este trabajo, completamos nuestra metodología de reparación [3, 6] para considerar y eliminar los errores de completitud. Empezamos analizando la relación entre los errores de completitud e introducimos dos órdenes ($\preceq_{inf}$ y $\preceq^{sup}$) que se corresponden con dos criterios diferentes de priorización u ordenación de los errores. Aprovechando este análisis formalizamos dos operaciones de reparación: *repairByDelete* la cual se apoya en el orden $\preceq_{inf}$ y *repairByInsert* la cual se apoya en el orden $\preceq^{sup}$. Con la utilización de estas operaciones, definimos dos estrategias que reducen la cantidad de acciones necesarias para reparar un sitio Web. Esto nos permite mejorar nuestro sistema de reparación, al tiempo que generar un sitio que cumple totalmente sus especificaciones.

Estructura del trabajo. El resto del trabajo está estructurado como sigue. La sección 2 resume algunas definiciones y notaciones preliminares. En la sección 3, recordamos un método simple de transformación de documentos XHTML/XML en términos de Herbrand, mostramos la noción de simulación utilizada para reconocer patrones dentro de un sitio Web y presentamos

el lenguaje de especificación de sitios Web. En la sección 4 formalizamos los errores de completitud que pueden ser detectados tras el proceso de verificación y presentamos dos operaciones que permiten su reparación. La sección 5 proporciona un análisis de la dependencia entre los errores de completitud, mientras en la sección 6 desarrollamos las estrategias de reparación de un sitio Web incompleto al tiempo que demostramos que la reparación no introduce nuevos errores. La sección 7 presenta las conclusiones del trabajo.

2 Preliminares

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 [12] (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 condiciones: $(i) X_i \neq X_j$, 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 [5, 14]).

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 $s \rightarrow_R^! t$) si $s \rightarrow_R^* t$ y t es irreducible. Decimos que un TRS R es *terminante*, si no existe una secuencia infinita $t_1 \rightarrow_R t_2 \rightarrow_R ....$ Un TRS R es *confluente* si, para todos los términos s, t_1, t_2, tal que $s \rightarrow_R^* t_1$ y $s \rightarrow_R^* t_2$, existe un término t s.t. $t_1 \rightarrow_R^* t$ y $t_2 \rightarrow_R^* t$. 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*.

3 Denotación de sitios Web

Sean dos alfabetos T y $\mathcal{T}ag$. Representaremos el conjunto T^* por $\mathcal{T}ext$. Llamaremos a un objeto $t \in \mathcal{T}ag$ elemento *tag*, y a un objeto $w \in \mathcal{T}ext$ elemento *text*. Las páginas Web se proporcionan con una estructura de árbol; de esta manera, pueden ser traducidas de forma directa a términos ordinarios de un álgebra de términos $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$ dada [2]. Note que los atributos tag XML/XHTML pueden considerarse elementos etiquetados comunes y, por tanto, traducidos de la misma forma que éstos. De esta manera, un *sitio Web* puede ser representado como un conjunto finito de términos *ground* $\{p_1 ... p_n\}$.

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 $s \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$ es llamado "plantilla de página Web" (*Web page template*).

En nuestra metodología usamos las plantillas de páginas Web para especificar patrones de páginas Web incompletas o ausentes en el sitio Web. Para mecanizar la detección de estos patrones dentro de un sitio Web, empleamos un mecanismo de simulación de árboles, que pasamos a describir en la siguiente sección. En las Figuras 1(a) y 1(b), se ve un ejemplo de transformación de una página Web a un elemento del álgebra de términos.

```
<book>                              book(
  <title>El Alquimista</title>       title(El Alquimista),
  <author>Coelho</author>            author(Coelho),          book(code(X),author(Y))
  <year>2002</year>                  year(2002),
  <code>PC02</code>                  code(PC02)
</book>                             )
          (a)                               (b)                        (c)
```

Fig. 1. Ejemplo de transformación de una página Web a un elemento del álgebra de términos, y una plantilla de página Web.

3.1 Simulación

La noción de *simulación* nos permite analizar y extraer la estructura parcial de una página Web sujeta a verificación. Las simulaciones se han utilizado en varios trabajos relacionados con la consulta, transformación y verificación de datos semiestructurados (ej. [1, 10, 13]). Para simplificar nuestra exposición, no consideramos la semántica de las etiquetas; esto requeriría introducir ontologías, lo cual queda fuera del alcance de este trabajo.

La noción de simulación, $\trianglelefteq$, es una adaptación de la relación de *embedding* de Kruskal [9], ignonardo la regla de *diving*[4] [15].

[4] 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.

Definition 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 se define como sigue:*

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

Dadas dos páginas Web s_1 y s_2, si $s_1 \trianglelefteq s_2$ se dice que s_1 *simula* (o *es embebida* o *reconocida* dentro de) s_2. También se dice que s_2 *embebe* s_1. Note que, en la Definición 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 $s_1 \not\trianglelefteq s_2$ si s_1 o s_2 contiene variables.

Un ejemplo de simulación se observa en la Figura 1: la plantilla de página Web de la Figura 1(c) puede ser reconocida dentro de la estructura del (fragmento de) página Web de la Figura 1(b).

3.2 Lenguaje de especificación

En esta sección, describiremos brevemente el lenguaje de especificación Web dado en [2]. Una especificación Web es una tripla (I_N, I_M, R), donde I_N e I_M son conjuntos finitos de reglas de corrección (*correctness rule*) y reglas de completitud (*completeness rule*), y R un conjunto de funciones auxiliares.

El conjunto I_N describe restricciones para detectar páginas Web erroneas (*correctNess rules*). Un análisis de optimización sobre los errores de corrección detectados se presentó en [7], y está fuera del alcance de este trabajo.

El tercer conjunto de reglas I_M especifica propiedades para detectar páginas Web incompletas/ausentes (*coMpleteness rules*). Una regla de completitud está definida como $l \rightharpoonup r \ \langle \mathtt{q} \rangle$, donde l y r son términos y $\mathtt{q} \in \{\mathtt{E}, \mathtt{A}\}$. Las reglas de completitud de una especifiación Web formalizan el requerimiento de que alguna información deba estar incluida en todas o algunas de las páginas del sitio Web. Los atributos $\langle \mathtt{A} \rangle$ y $\langle \mathtt{E} \rangle$ se usan para distinguir, respectivamente, reglas universales (*universal rule*) de las existenciales (*existential rule*), como explicamos mas adelante. La parte derecha de las reglas de completitud r puede contener funciones, las cuales están definidas en R y, algunos símbolos, pueden estar marcados por medio del símbolo $\sharp$. La información marcada en r se usa para seleccionar el subconjunto del sitio Web sobre el cual se verificará la condición formalizada por r.

Intuitivamente, la interpretación de una regla de completitud universal (resp. una regla de completitud existencial) w.r.t. un sitio Web W es la siguiente: si (una instancia de) l es "reconocida" en W, (una instancia de) la forma irreducible de r debe también ser "reconocida" en todas (resp. algunas de) las páginas Web que embeban (una instancia de) la estructura marcada de r. Una operación auxiliar *mark* es utilizada para obtener el subconjunto de páginas Web de W que se ajustan a la estructura de un término marcado r, por ejemplo, $mark(\sharp f(h(X)), \{p_1, p_2\}) = \{p_1\}$, donde $p_1 = f(m(a))$ y $p_2 = h(g(b))$.

3.3 Reescritura parcial

La reescritura parcial [2] nos permite extraer de una página Web s, usando una regla $l \rightharpoonup r$, una parte (subtérmino) de s que es simulada por alguna instancia *ground* de l, replazando

s por una instancia *ground* de r. Mas formalmente, sea s, $\mathtt{t} \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. Entonces, s *reescribe parcialmente* a $\mathtt{t}$ por medio de la regla $l \rightharpoonup r$ y la sustitución σ sii existe una posisición $u \in O_{\mathcal{T}ag}(s)$ s.t. (i) $l\sigma \trianglelefteq s_{|u}$, y (ii) $\mathtt{t} = \mathit{Reduce}(r\sigma,R)$, donde la función $\mathit{Reduce}(x, R)$ computa por reescritura de términos estándar la forma irreducible de x en R. Note que el contexto de la expresión reducible seleccionada $s_{|u}$ es omitido después de cada paso de reescritura. Por $\mathtt{s} \rightharpoonup_I \mathtt{t}$, denotamos que $\mathtt{s}$ se reescribe parcialmente a $\mathtt{t}$ usando alguna regla que pertenece al conjunto I.

4 Errores y acciones de reparación

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 denominaremos "errores de completitud" (*completeness error*). Estos errores representan la información inconsistente o ausente en el sitio Web.

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.

Estos tres tipos de errores (M, A, E) pueden ser detectados por reescritura parcial sobre las páginas Web y la especificación I_M.

Más formalmente tenemos la siguiente definición.

Definition 2 (completeness error). *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. $r\sigma = s_n$ para alguna $l \rightharpoonup r\langle \mathtt{q} \rangle \in I_M$.*
ii) *Existe una sustitución σ' s.t. $l'\sigma' = s_0$ para alguna $l' \rightharpoonup r'\langle \mathtt{q}' \rangle \in I_M$.*
iii) *Existe $p \in W$ s.t. $s_0 \trianglelefteq p$.*
iv) *$P = \{p \mid p \in \mathit{mark}(r, W)$ y $s_n \not\trianglelefteq p\}$.*
v) *$c = (M$ si $P = \emptyset)$ o $(\mathtt{q}$ si $P \neq \emptyset)$.*

En la definición 2, $s_0 \rightharpoonup^*_{I_M} \rightharpoonup s_{n-1} \rightharpoonup_{I_M} s_n$ 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.

Con el objetivo de reparar un sitio Web que contiene errores, en [3] definimos tres acciones reparadoras básicas de corrección que permiten eliminar la inconsistencia y/o añadir la información ausente. Las acciones consideradas son las siguientes:

- **delete**(p, t) elimina todas las ocurrencias del término t en de la página Web p, y devuelve la página Web modificada.

– **insert**(p, w, t) modifica la página Web p añadiendo el término t en $p_{|w}$.
– **add**(p, W) añade la página Web p al sitio Web W.

Note que las acciones reparadoras *add* e *insert* introducen nueva información en el sitio Web que puede contener/provocar nuevos errores. Por ello, es importante restringir la información que se puede añadir. Del mismo modo, la ejecución de la acción *delete* puede inducir nuevos errores de completitud. Para prevenir estas situaciones, en [3] formalizamos las propiedades *safe* y *acceptable* que garantizan el comportamiento seguro de estas acciones.

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

A lo largo de este trabajo, asuminos que la aplicación de las acciones reparadoras es siempre *acceptable*. Un sitio Web está reparado si $\mathbb{E}_M = \phi$.

4.1 Reparando errores de completitud

En esta sección se formulan las operaciones para reparar un sitio Web que contenga errores de completitud. Sin pérdida de generalidad, se asume que el sitio Web W es incompleto pero correcto w.r.t. una especificación Web (I_N, I_M, R). Esto se justifica por la posibilidad de asumir el haber ejecutado previamente la metodología de reparación de la corrección de [3]. Tal suposición permite diseñar una metodología reparadora la cual "completa" el sitio Web sin introducir información incorrecta. Como dijimos en la sección anterior, cualquier error $e \in \mathbb{E}_M(W)$ puede ser reparado añadiendo la información ausente o eliminando el dato que lo produjo. De igual forma, consideramos *acceptable* a cualquier acción que se ejecute, lo cual se puede comprobar fácilmente ejecutando la metodología de verificación de [2].

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

4.2 Añadiendo información

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

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

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

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

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

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

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

Note que la información añadida para reparar un error podría casualmente ser la misma que sirva para reparar otro error. Postponemos a la Sección 5 el análisis de la posible relación entre los errores.

4.3 Eliminando información incompleta

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 es eliminar aquella información del sitio Web que causa el error de completitud sin necesidad de remplazar la nueva por información que satisfaga las especificaciones. El tratamiento de la información incompleta es independiente del tipo de error (M, A, E) que estemos analizando; por lo tanto, la información ausente se computa de la misma manera para los tres tipos de errores.

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

Definition 3 (*repairByDelete*)**.** *Dado un sitio Web W y un error de completitud $e \equiv (s_0 \rightharpoonup \ldots \rightharpoonup s_n, P, q)$, el sitio Web W se transforma aplicando acciones de borrado de la siguiente manera.*

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

En otras palabras, en la Definición 3 se elimina de todas las páginas Web la ocurrencia del término s_0 que inicia la secuencia de reescritura. Note que, si en la secuencia de reescritura de dos errores tenemos el mismo término inicial, es posible que la reparación de un error corrija de manera automática a otro.

5 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.

Definition 4. *(orden inducido por los inferiores) 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,*

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

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 $e' \not\equiv e$.*

Definition 5. *(orden inducido por los superiores) 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 4 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, 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) w.r.t. 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 $\preceq_{inf}$.

Proposition 1. *Sea* W *un sitio Web, y* $e_i \in \mathbb{E}_M(W), i = 1 \ldots m$*, 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 4 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 esta proposición 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 es debido 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 un error universal es reparado, es posible que páginas que pertenezcan a otro error universal inferiror sean reparadas, y por lo dicho anteriormente, no pertenecen más al error.
- Una vez aplicada alguna acción reparadora en una página, no es necesario volver 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 1 y el Algoritmo 1, permiten una optimización obvia del marco de reparación, que formalizamos en las estrategias de reparación presentadas a continuación.

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

Require:
$\mathbb{E} = e_i \in \mathbb{E}_M(W), i = 1, \dots m$, and $[e_1 \preceq^{sup} \dots \preceq^{sup} e_m]$
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
```

6 Estrategias de reparación

Cómo explicamos en la Sección 4, es posible reparar un error de completitud por la ejecución de alguna acción reparadora. Por (e, a) denotamoa 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.

Definition 6 (estrategia de reparación). *Sea W un sitio Web y sea $\mathbb{E}_M(W) = \{e_1, \dots, e_n\}$ el conjunto de errores de completitud en W. Una* estrategia de reparación *para W es la secuencia $[(e_1, a_1), \dots, (e_n, a_n)]$, donde $a_1, \dots, a_n$ son acciones reparadoras s.t.*

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

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

En la Sección 5 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.

6.1 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 1, 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.

Definition 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* $\mathbb{RDA}$-strategy*) que permite reducir las acciones de eliminación para W es*

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

La Definición 7 determina la estrategia $\mathbb{RDA}$-*strategy*, la cual consiste en reparar todos los errores minimales w.r.t. la relación $\preceq_{inf}$ de un sitio Web. Esto nos lleva a la siguiente proposición, que establece que la $\mathbb{RDA}$-*strategy* permite obtener un sitio Web W libre de errores reparando sólo un subconjunto de los errores de $\mathbb{E}_M(W)$.

Proposition 2. *Sea W un sitio Web y sea $\mathbb{E}_M(W)$ el conjunto de errores de completitud de W. Entonces, (i) la estrategia* $\mathbb{RDA}$-strategy *obtiene un sitio Web libre de errores; (ii) la cantidad de acciones reparadoras ejecutadas por la estrategia* $\mathbb{RDA}$-strategy *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 1, e será reparado de manera automática sin necesidad de ejecutar una acción reparadora para e.

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

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

La secuencia $\mathbb{C}_{\mathbb{E}_M}$ está ordenada por la cardinalidad de los conjuntos que la componen. Denotaremos por $\mathbb{C}_{\mathbb{E}_M}(i)$ 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

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

Definition 8 (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* $\mathbb{RIA}$-strategy*) para W es*

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

La estrategia de la Definición 8 ejecuta el procedimiento $repairByInsert$ en cada conjunto de la partición de $\mathbb{E}_M$.

Proposition 3. *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* $\mathbb{RIA}$-strategy *obtiene un sitio Web libre de errores; (ii) la cantidad de acciones reparadoras ejecutadas por la estrategia* $\mathbb{RIA}$-strategy *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 $\mathbb{RIA}$-*strategy* 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*.

Example 1. 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 4 y 5 respectivamente, establecen los siguientes subconjuntos de errores

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

A continuación describiremos cómo aplicar las estrategias *reduce-delete-actions* y *reduce-insertion-actions* definidas en las secciones 6.1 y 6.2 respectivamente.

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

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

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

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

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

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

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

$$\Gamma(\mathbb{E}_M) = \{m_1, m_2, m_3\}$$
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 $m \in \Gamma(\mathbb{E}_M)$:

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

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

7 Conclusiones

Encontrarse con información incompleta o indeseada en la Web es muy frecuente, por eso la administración y mantenimiento de la Web es un problema abierto y urgente. En este trabajo, completamos la metodología de reparación semiautomática de sitios Web erróneos presentada en [3] con el tratamiento y eliminación de los errores de completitud. Para ello, establecemos dos relaciones de órden parcial ($\preceq_{inf}$ y $\preceq^{sup}$) sobre los errores de completitud detectados. Con estos órdenes, realizamos un análisis entre la dependencia existente entre los errores, lo cual permitió definir dos estrategias de reparación: *reduce-delete-actions* y *reduce-insertion-actions*. La primera focaliza la reducción de la cantidad de información que es necesaria eliminar para obtener un sitio Web libre de errores con respecto a su especificación. La segunda minimiza la información que es necesaria 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 reaparación de un sitio Web; esto es debido a que el usuario debe de reparar una cantidad menor de errores para obtener un sitio Web correcto y completo. Como trabajo futuro, pensamos en la integración de estas estrategias de reparación en nuestro sistema WebVerdi-M [4] (disponible en http://www.dsic.upv.es/users/elp/webverdi-m/). También hemos comenzado a explorar la posibilidad de utilizar la interpretación abstracta para reducir el coste de la verificación y la reparación.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.
2. M. Alpuente, D. Ballis, and M. Falaschi. Rule-based Verification of Web Sites. *Software Tools for Technology Transfer*, 8:565–585, 2006.
3. M. Alpuente, D. Ballis, M. Falaschi, and D. Romero. 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)*, pages 31–40. IEEE Computer Society Press, 2006.
4. M. Alpuente, D. Ballis, M. Falaschi P. Ojeda, and D. Romero. A Fast Algebraic Web Verification Service. In *Proc. of First Int'l Conf. on Web Reasoning and Rule Systems (RR 2007)*, 2007. to appear.
5. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
6. D. Ballis. *Rule-based Software Verification and Correction*. PhD thesis, University of Udine and Technical University of Valencia, 2005.
7. D. Ballis and D. Romero. 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*, pages 11–19. IEEE Computer Society Press, 2007.
8. L. Bertossi and J. Pinto. Specifying Active Rules for Database Maintenance. In G. Saake, K. Schwarz, and C. Türker, editors, *Transactions and Database Dynamics, 8th Int'l Workshop on Foundations of Models and Languages for Data and Objects*, volume 1773 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 1999.
9. M. Bezem. *TeReSe, Term Rewriting Systems*, chapter Mathematical background (Appendix A). Cambridge University Press, 2003.
10. F. Bry and S. Schaffert. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *Proc. of the Int'l Conference on Logic Programming (ICLP'02)*, volume 2401 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
11. L. Capra, W. Emmerich, A. Finkelstein, and C. Nentwich. XLINKIT: a Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technology*, 2(2):151–185, 2002.
12. N. Dershowitz and D. Plaisted. Rewriting. *Handbook of Automated Reasoning*, 1:535–610, 2001.
13. M. F. Fernandez and D. Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In *Proc. of Int'l Conf on Data Engineering (ICDE'98)*, pages 14–23, 1998.
14. J.W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I, pages 1–112. Oxford University Press, 1992.
15. M. Leuschel. Homeomorphic Embedding for Online Termination of Symbolic Methods. In T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors, *The Essence of Computation*, volume 2566 of *LNCS*, pages 379–403. Springer, 2002.
16. E. Mayol and E. Teniente. A Survey of Current Methods for Integrity Constraint Maintenance and View Updating. In *Proc. of Advances in Conceptual Modeling: ER '99*, volume 1727 of *Lecture Notes in Computer Science*, pages 62–73. Springer, 1999.
17. C. Nentwich, W. Emmerich, and A. Finkelstein. Consistency Management with Repair Actions. In *Proc. of the 25th International Conference on Software Engineering (ICSE'03)*. IEEE Computer Society, 2003.
18. Jan Scheffczyk, , Uwe M. Borghoff Peter Rödig, and Lothar Schmitz. S-dags: Towards efficient document repair generation. In *Proc. 2nd Int. Conf. on Computing, Communications and Control Technologies*, volume 2, pages 308–313, 2004.
19. Jan Scheffczyk, Uwe M. Borghoff, Peter Rödig, and Lothar Schmitz. Consistent document engineering: formalizing type-safe consistency rules for heterogeneous repositories. In *Proc. of the 2003 ACM Symposium on Document Engineering (DocEng '03)*, pages 140–149. ACM Press, 2003.
20. Jan Scheffczyk, Peter Rödig, Uwe M. Borghoff, and Lothar Schmitz. Managing inconsistent repositories via prioritized repairs. In *Proc. of the 2004 ACM Symposium on Document Engineering (DocEng '04)*, pages 137–146. ACM Press, 2004.