Rewriting-based Repairing Strategies for XML Repositories[☆]

María Alpuente[a], Demis Ballis[b], Moreno Falaschi[c], Francisco Frechina[a], Daniel Romero[a]

[a]*DSIC-ELP, Universitat Politècnica de València,*
Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain.
[b]*Dipartimento di Matematica e Informatica,*
Via delle Scienze 206, 33100 Udine, Italy.
[c]*Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche,*
Pian dei Mantellini 44, 53100 Siena, Italy.

Abstract

Keeping XML data in a consistent state w.r.t. both structure and content is a burdensome task. To maintain the consistency of ever-larger, complex XML repositories, suitable mechanisms that are able to fix every possible inconsistency are needed. In this article, we present a methodology for semi-automatically repairing faulty XML repositories that can be integrated on top of an existing rewriting-based verification engine. As a formal basis for representing consistency criteria, we use a rule-based description formalism that is realized in the language Maude. Then, starting from a categorization of the kinds of errors that can be found during the verification process, we formulate a stepwise transformation procedure that achieves correctness and completeness of the XML repository w.r.t. its Maude formal specification while strictly observing the structure of the XML documents. With the aim of increasing the level of automation of our repair methodology, we also define two correction strategies and two completion strategies that reduce either the amount of information to be changed or the number of repair actions to be executed in order to deliver an XML repository that is both correct and complete. Finally, we describe a prototype implementation of the repairing tool, which we use for an experimental evaluation of our method with good results.

Keywords: repairing strategies, XML consistency, rule-based techniques

[☆]This work has been partially supported by the EU (FEDER) and the Spanish MEC project ref. TIN2010-21062-C02-02, and by Generalitat Valenciana ref. PROMETEO2011/052. This work was carried out during the tenure of D. Ballis' ERCIM "Alain Bensoussan" Postdoctoral Fellowship. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 246016. F. Frechina was supported by FPU-ME grant AP2010-5681 and D. Romero by FPI-MEC grant BES-2008-004860.

Email addresses: `alpuente@dsic.upv.es` (María Alpuente), `demis.ballis@uniud.it` (Demis Ballis), `moreno.falaschi@unisi.it` (Moreno Falaschi), `ffrechina@dsic.upv.es` (Francisco Frechina), `dromero@dsic.upv.es` (Daniel Romero)

1. Introduction

The semi-structured nature, frequent updates, and the typically large number of document sources and contributing authors/applications to XML repositories render the consistency and quality ensuring of XML-based documentation a tedious and error-prone task. Actually, Web systems are very often collaborative applications in which many users freely contribute to update their contents (e.g. wikis, blogs, social networks, . . .). In this scenario, the task of keeping data correct and complete is particularly arduous, because of the very poor control over the content update operations which may easily lead to data inconsistency problems.

A lot of research work has been invested in consistency management and repair of software applications and databases, but similar technologies for XML-based systems are much less mature. In [1], a repair framework for inconsistent distributed documents that complements the tool xlinkit [2] is presented. The main contribution is the semantics that maps the first-order logic language of xlinkit (without interpreted functions) to a catalogue of repairing actions that can be used to interactively correct rule violations. Consistency rules are expressed in first-order logic with embedded XPath predicates and may refer to external resources that can be mapped to XML representations such as relational databases. If rules are violated, suggestions for repairing the detected errors are generated and presented to the user (Nentwich et al., 2003). However, the system does not predict whether a repair action can cause new errors to appear. Also, xlinkit is not designed to be used by authors that do not have detailed knowledge about XML processing. The xlinkit engine has been integrated into the commercial software "messageAUTOMATION validator" [3], which offers a graphical rule editor and a means for rule testing and management (Message Automation Ltd, 2010). Similarly, in [4, 5], an extension for the tool CDET [6] is presented. This extension includes a mechanism to remove inconsistencies from sets of interrelated documents, which is done by first generating direct acyclic graphs (DAGs) that represent the relations among the documents and then deriving suitable repairs directly from the DAGs. The formal basis is full first-order logics that is interpreted over a language that relies on Haskell. Temporal rules are supported and the interference of repairs is not completely neglected. Both approaches [1] and [4] rely on basic techniques that are borrowed from the field of active databases [7]. Current research in this field focuses on the derivation of active rules that automatically fire repair actions leading to a consistent state after each update [8].

Coelho et al. [9, 10, 11] use an extension of Prolog called XCentric to check and repair the syntactic and semantic properties for the content of XML-based Web sites. The XML Web document is translated into a temporary document that is composed of logical terms that correspond to the XML tags in the original document. A sequence of checking and repairing rules is applied on the translated document to verify the semantics of its content. The verified document is then translated back to its original representation in XML. The main application of this tool is to verify the content of collaborative Web sites such as Wikipedia [9]. The framework was first implemented in [10] and then improved in [9, 11].

In [12], a methodology and a tool are proposed to suggest repairs to Web sites that violate some requirements that are given in the form of Web rules and expressed in a fragment of Xcerpt. The methodology consists of translating these Web rules into abductive logic programs with constraints and processing the transformed programs by means of a general-purpose proof procedure called CIFF. The repairing tool, called CIFFWEB, consists of CIFF together with the translation from rules to programs and from Web sites to a suitable logical format.

In the Verdi project [13, 14, 15, 16, 17, 18], a rule-based, formal specification language has been used to define syntactic/semantic properties of XML repositories and XML-based Web sites. The key idea behind this approach is to exploit the power of term rewriting [19] and Rewriting Logic (RWL) [20] to support the specification and efficient manipulation of XML/HTML documentation in a natural and effective way. Rewriting Logic is a very general logical and semantic framework that is particularly suitable for formalizing highly concurrent, complex systems (e.g., biological systems and Web systems [21]). RWL is efficiently implemented in the high-performance system Maude [22]. In the Verdi project, a rewriting-based verification facility, which was developed in Maude, identifies the requirements not fulfilled by the XML repository and helps in repairing errors by finding wrong, incomplete or missing XML contents. Consistency of the original XML documents is checked with respect to both syntactic and semantic formal descriptions. This approach has several advantages for document verification. We not only extract the erroneous data but we also identify its source in the documents, a feature which is necessary for error tracing and repairing. The verification engine was endowed with a semi-automatic repairing methodology in [16] and a preliminary implementation of the repairing methodology was described in [23].

While the errors in a given XML repository are often deeply interrelated, the rewriting-based repair framework of [16] does not explore the relation/dependencies among the identified errors. Such an analysis can be a potential source of optimization and may increase the level of automation of the repair system because the correction of a given bug may lead to an automatic fix for a “related” bug without the need to execute a specific repair action. This idea was first explored in the correction strategies also defined in [23].

In recent years, the rule-based approach has experienced growing popularity in XML applications [24]. Nevertheless, as far as we know, there has been little exploration of the rewriting-based techniques for the repairing of XML repositories to date. Actually, we only know of one previous rewriting-based approach for XML processing and transformation: a term rewriting implementation is provided in [25] for (a fragment of) XSLT, the rule-based language designed by W3C for the transformation of XML documents. A related rewriting-based approach is [26], which defines HTML transformations with the aim of improving Web applications by cleaning up syntax, reorganizing frames, or updating to new standards. The work in [24] defines a rule-based approach to XML processing and Web reasoning that uses an extension of logic programming using unranked signatures (flexible arity, i.e., variadic symbols) and advanced rule-based programming features for hedge (aka unranked term sequence) transformations, strategies, and regular constraints. A Rewriting Logic approach to the formal specification and verification of dynamic Web applications can be found in [27].

This article is a revised and extended version of our methodology for repairing faulty XML repositories in [16, 23] that provides further conceptual and technical content. The main contributions of this work can be summarized as follows:

- A rewriting-based framework for document verification and repair is presented where the emphasis is placed on providing strategies to reduce the complexity and to increase the quality of the repaired documentation. The rule-based approach supports the concise definition of repairing rules for document formats based on XML or HTML.

- We complement our methodology by defining two correction strategies and two completion strategies that increase the level of automation of our repair method. Specifically, the proposed strategies optmize both the amount of information to be changed and the number of repair actions to be executed in a faulty XML repository to make it both correct and complete. In both cases, it is worth noting that the number of errors that we need to correct in order to get a repaired XML repository W is much lower than the total number of errors that occur in W. Consequently, employing these strategies may significantly improve the performance of the repair process. To our knowledge, no repair system supports this kind of optimization based on repairing strategies, which can lead to a faster and simpler correction of the faulty repository.

- We also provide a new, totally redesigned implementation of the proposed technique in order to: (i) provide automatic support for the repairing strategies, (ii) improve the performance of the tool, and (iii) make the system easier to use and extend. The implemented tool, called WifiX, together with a set of examples and its XML API is publicly available at `http://safe-tools.dsic.upv.es/repairing`. An on-line Java Web client that interacts with the Web verification and repair service is also available.

Plan of the Paper. The rest of this article is organized as follows. Section 2 summarizes some preliminary definitions and notations about term rewriting systems that are needed in our formulation. In Section 3, we first briefly recall the rewriting-based XML verification framework of [15]. We also categorize the different kinds of errors that can be identified in a faulty repository and that are the input for the repair process. In Section 4, we provide a catalogue of repair actions that can be used to define some basic repair techniques. These techniques allow correctness as well as completeness errors to be fixed through suitable delete/insert actions which modify the content of the XML repository under examination. In Section 5, we carry out a systematic analysis on the relations among correctness errors, which we exploit to formalize two optimized, semi-automatic correction strategies that deliver a correct XML repository: 1) the $\mathbb{NAR}$ strategy (correctio$\mathbb{N}$ with $\mathbb{A}$ction $\mathbb{R}$eduction), which allows the number of repair actions that are executed to be reduced, and 2) the $\mathbb{NDR}$ strategy (correctio$\mathbb{N}$ with $\mathbb{D}$ata $\mathbb{R}$eduction), which reduces the amount of information to be changed/removed in order to correct the XML repository. Similarly, with regard to completeness, in Section 6 we formalize two additional optimized, completion strategies that deliver a complete XML repository: 3) the $\mathbb{MDR}$ strategy (co$\mathbb{M}$pletion with $\mathbb{D}$eletion $\mathbb{R}$eduction), which reduces the number of deletion repair actions that are needed to complete the XML repository, and 4) the $\mathbb{MAR}$ strategy (co$\mathbb{M}$pletion with $\mathbb{A}$ddition $\mathbb{R}$eduction), which allows the number of addition/insertion repair actions that are executed to be reduced. In all cases, it is worth noting that the number of errors that we need to consider to get a repaired XML repository W is much lower than the total number of errors that occur in W. Consequently, employing these strategies guarantees a more efficient repair. Section 7 describes the repairing tool WifiX and its service-oriented architecture, and reports on some experiments that we have conducted. Finally, Section 8 concludes.

2. Preliminaries

Term rewriting systems provide an adequate computational model for functional languages. In the sequel, we follow the standard framework of term rewriting (see [28]).

By $\mathcal{V}$, we denote a countably infinite set of variables, and Σ denotes a set of function symbols, or *signature*. We consider variadic signatures as in [29] (i.e., signatures in which symbols have an unbounded arity, that is, they may be followed by an arbitrary number of arguments). $\tau(\Sigma, \mathcal{V})$ and $\tau(\Sigma)$ denote the *non-ground term algebra* and the *term algebra* built on $\Sigma \cup \mathcal{V}$ and Σ. Terms are viewed as labeled trees in the usual way. Positions are represented by sequences of natural numbers that denote an access path in a term. The empty sequence Λ denotes the root position. By notation $w_1.w_2$, we denote the concatenation of position w_1 and position w_2. Positions are ordered by the prefix ordering, that is, given the positions w_1 and w_2, $w_1 \leq w_2$ if there exists a position x such that $w_1.x = w_2$. Given $S \subseteq \Sigma \cup \mathcal{V}$, $O_S(t)$ denotes the set of positions of a term t that are rooted by symbols in S. Moreover, for any position x, $\{x\}.O_S(t) = \{x.w \mid w \in O_S(t)\}$. $t_{|u}$ is the subterm at the position u of t. $t[r]_u$ is the term t with the subterm rooted at the position u replaced by r. We say that a term t is *ground*, if no variables occur in t. A term t is *linear*, if t does not contain multiple occurrences of the same variable. Syntactic equality between objects is represented by $\equiv$. Given a set or a sequence of elements S, by $|S|$ we denote the number of elements in S.

A *substitution* $\sigma \equiv \{X_1/t_1, X_2/t_2, \ldots\}$ is a mapping from the set of variables $\mathcal{V}$ to the set of terms $\tau(\Sigma, \mathcal{V})$ which is almost everywhere equal to the identity. By *id* we denote the *identity* substitution. The application of a substitution σ to a term t, denoted $t\sigma$, is defined by induction on the structure of terms:

$$t\sigma = \begin{cases} x\sigma & \text{if } t = x \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

In the second case of this denition, $n = 0$ is allowed: in this case, f is a constant symbol and $f\sigma = f$.

Given a substitution σ, the *domain* of σ is the set $Dom(\sigma) = \{X | X\sigma \not\equiv X\}$. Given two substitutions σ_1 and σ_2, such that $Dom(\sigma_2) \subseteq Dom(\sigma_1)$, by σ_1/σ_2 we define the substitution $\{X/t \in \sigma_1 \mid X \in Dom(\sigma_1) \setminus Dom(\sigma_2)\} \cup \{X/t \in \sigma_2 \mid X \in Dom(\sigma_1) \cap Dom(\sigma_2)\} \cup \{X/X | X \not\in Dom(\sigma_1)\}$.

By $\mathcal{V}ar(s)$ we denote the set of variables that occurs in the syntactic object s.

A *term rewriting system* (TRS for short) is a pair (Σ, R), where Σ is a signature and R is a finite set of reduction (or rewrite) rules of the form $\lambda \to \rho$, λ, $\rho \in \tau(\Sigma, \mathcal{V})$, $\lambda \not\in \mathcal{V}$, and $\mathcal{V}ar(\rho) \subseteq \mathcal{V}ar(\lambda)$. We will often write just R instead of (Σ, R). Sometimes, we denote the signature of a TRS (Σ, R) by Σ_R.

A rewrite step is the application of a rewrite rule to an expression. A term s *rewrites* to a term t via $r \in R$, $s \to_r t$ (or $s \to_R t$), if there is a position $u \in O_\Sigma(s)$, $r \equiv \lambda \to \rho$, and a substitution σ such that $s_{|u} \equiv \lambda\sigma$ and $t \equiv s[\rho\sigma]_u$. When no confusion can arise, we will omit any subscript (i.e. $s \to t$). A term s is an *irreducible form* (or *normal form*) w.r.t. R if there is no term t s.t. $s \to_R t$. t is the irreducible form of s w.r.t. R (in symbols $s \to^!_R t$) if $s \to^*_R t$ and t is irreducible.

We say that a TRS R is *terminating* if there is no infinite rewrite sequence $t_1 \to_R t_2 \to_R \ldots$ A TRS R is *confluent* if, for all terms s, t_1 and t_2 such that $s \to^*_R t_1$ and $s \to^*_R t_2$, there exists a term t s.t. $t_1 \to^*_R t$ and $t_2 \to^*_R t$. When R is terminating and confluent, it is called *canonical*. In canonical TRSs, each input term t can be reduced to a unique irreducible form.

We say that the equation $s = t$ *holds* in a canonical TRS R, if there exists an irreducible form $z \in \tau(\Sigma, \mathcal{V})$ w.r.t. R such that $s \to^!_R z$ and $t \to^!_R z$. Inequalities are denoted by expressions of the form $s \neq t$. An inequality $s \neq t$ *holds* in a canonical TRS R iff the equation $s = t$ does not hold in R.

Given a sequence $c_1, \ldots c_m$, where each c_i, $i = 1, \ldots, m$, is either an equation or an inequality, we say that $c_1, \ldots c_m$ *holds* in a canonical term rewriting system R w.r.t. a substitution σ, if each $c_i\sigma$, $i = 1, \ldots, m$, holds in R.

3. Rewriting-based XML Verification

In this section, we briefly recall the formal verification methodology proposed in [15], which allows us to detect erroneous or missing data in XML repositories that model the information content of Web systems. First, we show that XML repositories can be naturally interpreted as sets of terms over a suitable term algebra. Then, we present the rule-based specification language that we devised to formally verify XML repositories.

3.1. Modeling XML repositories

XML documents are provided with a tree-like structure that allows them to be straightforwardly encoded into ordinary terms of a suitable term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$[1] More precisely, let us consider a finite set of 0-ary symbols T and a finite set of variadic symbols $\mathcal{T}ag$. We denote the set of all finite sequences of elements over T by $\mathcal{T}ext$, and assume that the sets $\mathcal{T}ag$ and $\mathcal{T}ext$ are disjoint. An XML document p is any term of $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$ that can be generated by the following BNF-like grammar [2]:

$$\begin{array}{rcll} \langle \mathit{XMLDoc} \rangle & := & t(\langle \mathit{XMLDocList} \rangle) \ \ | \ \ w & \text{with } w \in \mathcal{T}ext, t \in \mathcal{T}ag \\ \langle \mathit{XMLDocList} \rangle & := & \langle \mathit{XMLDoc} \rangle \ , \ \langle \mathit{XMLDocList} \rangle \ \ | \ \ \epsilon & \end{array}$$

Througout the paper, XML documents of the form $t()$ are simply denoted by t. *XML repositories* are finite sets of XML documents.

Example 3.1

Figure 1 specifies an XML repository as a set of ground terms. This repository includes the data of a research group such as member affiliations, scientific publications, research projects, teaching information, and member personal data.

Similarly, to represent templates of XML documents that share a common structure, we will also consider non-ground XML documents in the term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, which we call *XML document templates.*

3.2. The rule-based specification language

An XML specification is a triple (R, I_{CN}, I_{CM}), where R, I_{CN}, and I_{CM} are finite sets of rules. The set R contains the definition of some auxiliary functions that the user would like to provide (e.g., string processing, arithmetic, and boolean operators). R is formalized as a canonical term rewriting system, which is handled by standard rewriting [28].

The second set I_{CN} describes constraints for detecting erroneous XML documents (*CorrectNess rules*). A correctness rule has the following form: $\lambda \rightharpoonup \mathit{error} \mid C$, with $Var(C) \subseteq Var(\lambda)$, where λ is a linear term, *error* is a reserved constant, and C is a condition —i.e. a (possibly empty) finite sequence of equations and inequalities $c_1, \ldots, c_n$. We denote the *empty* condition by $\emptyset$.

Roughly speaking, when an instance $\lambda\sigma$ of λ is recognized inside an XML document and the condition C holds in the TRS R w.r.t. σ, then a correctness error is detected. Given a correctness rule $r_{CN} = \lambda \rightharpoonup \mathit{error} \mid C$, $\lambda \rightharpoonup \mathit{error}$ is called the *unconditional part* of r_{CN}.

[1] To keep the framework simple, we consider XML tag attributes as common tagged elements, and hence translated in the same way.

[2] Note that symbol ϵ in the BNF-like grammar denotes the empty string.

```
(P1)    {members(member(name(mario),surname(rossi),status(professor)),
                member(name(franca),surname(bianchi),status(technician)),
                member(name(giulio),surname(verdi),status(student)),
                member(name(ugo),surname(blu),status(professor))
               ),
(P2)     person(fullname(mariorossi),phone(3333),status(professor)),
(P3)     person(fullname(francabianchi),status(technician),phone(5555),
               email(bianchi@myresearchgroup.org)),
(P4)     person(fullname(annagialli),status(professor), email(gialli@gmail.com),
                phone(4444),teaching(course(algebra))),
(P5)     pubs(pub(name(ugo),surname(blu),title(pubtitle1),blink(year(2003))),
             pub(name(anna),surname(gialli),title(pubtitle2),year(2002))),
(P6)     projects(project(pname(A1),grant1(1000),grant2(200),
                          total(1100),coordinator(fullname(mariorossi))),
                  project(pname(B1),grant1(2000),grant2(1000),
                          projectleader(surname(gialli),name(anna)),
                          total(3000)))}
```

Figure 1: An example of an XML repository for a research group

It is worth noting that the expressiveness of correctness rules is at least comparable with the expressive power of regular expression types. Indeed, the rule condition C may include calls to functions which are specified by using the term rewriting system formalism (which provides an adequate, Turing-complete, computational model for first-order functional languages). For instance, the validation of an XML document p w.r.t. a given XML type (modeled by, e.g., a DTD or a more powerful XML Schema specification) can be formalized by a correctness rule of the form

$$p \rightharpoonup \mathit{error} \mid \mathit{validate}(p) = false$$

where *validate* is a function, whose TRS specification implements the desired validation algorithm.

The third set of rules I_{CM} specifies some requirements that signal incomplete/missing XML documents (*CoMpleteness rules*). A completeness rule is defined as $\lambda \rightharpoonup \rho\ \langle q \rangle$, where λ and ρ are terms, λ is linear, and $q \in \{E, A\}$. A Completeness rule $\lambda \rightharpoonup \rho\ \langle q \rangle$ specifies that an instance of the right-hand side ρ must be included in all documents ($\langle A \rangle$) or some documents ($\langle E \rangle$) of the XML repository, whenever an embedding of the left-hand side λ is detected in some XML document. We use attributes $\langle A \rangle$ and $\langle E \rangle$ to distinguish "universal" from "existential" rules.

Sometimes, we may be interested in checking a given completeness property only on a subset P of the whole repository. For this purpose, some symbols in the right-hand sides of the rules can be marked by means of the fresh constant symbol $\sharp$. Marking information of a given rule r is used to compute the subset P on which we want to check the property formalized by r. More specifically, rule r is executed on all and only the Web pages that embed the marked structure of r. For instance, consider the XML repository $W = \{f(a,b,c), f(b), f(c,a)\}$ and the completeness rule $r_{CM} = f(a) \rightharpoonup \sharp f(\sharp a, b)\langle A \rangle$. In this case, the universal rule r_{CM} is executed only on the set $P = \{f(a,b,c), f(c,a)\}$ that contains those XML documents of W embedding the marked term $f(a)$.

Given a completeness rule $r_{CM} = \lambda \rightharpoonup \rho\ \langle q \rangle$, $\lambda \rightharpoonup \rho$ is called the *unquantified part* of r_{CM}.

Example 3.2

Consider the XML specification that consists of the following completeness and correctness rules together with a term rewriting system that defines the string concatenation function `++`, the relational operator $\leq$, the arithmetic operators `+` and `*` for natural numbers, and the boolean function $\mathtt{inRegExp(s,e)}$ that checks whether a string `s` belongs to the regular language denoted by the regular expression $\mathtt{e}$[3]. That is:

r_1) $\mathtt{member(name(X),surname(Y))} \rightharpoonup \sharp\mathtt{person(fullname(X} \mathbin{+\!\!+} \mathtt{Y),status)}\ \langle \mathtt{E} \rangle$
r_2) $\mathtt{person(status(professor))} \rightharpoonup \sharp\mathtt{person(}\sharp\mathtt{status(}\sharp\mathtt{professor),teaching)}\ \langle \mathtt{A} \rangle$
r_3) $\mathtt{pub(name(X),surname(Y))} \rightharpoonup \sharp\mathtt{member(name(X),surname(Y))}\ \langle \mathtt{E} \rangle$
r_4) $\mathtt{courselink(url(X),urlname(Y))} \rightharpoonup \mathtt{coursedocument(title(Y))}\ \langle \mathtt{E} \rangle$
r_5) $\mathtt{email(X)} \rightharpoonup \mathtt{error} \mid \mathtt{inRegExp(X,[:Text:]^{+}@myresearchgroup.org)} = \mathtt{false}$
r_6) $\mathtt{project(grant1(X),grant2(Y),total(Z))} \rightharpoonup \mathtt{error} \mid \mathtt{X} + \mathtt{Y} \neq \mathtt{Z}$
r_7) $\mathtt{project(grant1(X),grant2(Y))} \rightharpoonup \mathtt{error} \mid \mathtt{X} \neq \mathtt{Y} * \mathtt{2}$
r_8) $\mathtt{total(Z)} \rightharpoonup \mathtt{error} \mid \mathtt{500000} \leq \mathtt{Z} = \mathtt{true}$

This XML specification models a number of properties for the XML repository of Figure 1. Rules r_1–r_4 are completeness rules, while r_5–r_8 are correctness rules.

The first rule formalizes the following property: if there is an XML document that contains group member information, then for each member, an XML personal record should exist that contains (at least) the full name and the status of this member. The full name is computed by concatenating the name and the surname strings by means of the `++` function.

The second rule states that, whenever a personal record of a professor is recognized, that record must also include some teaching information. The rule is universal since it must hold for each professor personal record.

The third rule specifies that, whenever there exists an XML document that contains information about scientific publications, each author of a publication should be a member of the research group.

The fourth rule formalizes the broken link property, that is, for each link to a course, there must be a document that describes that course.

The fifth rule specifies invalid email addresses by using the regular expression

$$[:\mathtt{Text}:]^{+}\mathtt{@myresearchgroup.org},$$

which identifies the regular language that contains all the strings of the form `text@myresearchgroup.org`, where `text` is a string built over the set $\mathcal{T}ext$.

[3] Regular languages are represented by means of the usual Unix-like regular expressions syntax.

The last three rules (r_6, r_7, and r_8) state that, for each research project, the total project budget must be equal to the sum of the grants (r_6), the first grant must double the second one (r_7), and the total budget must be less than 500000 euros (r_8).

The error diagnoses are carried out by running XML specifications on XML repositories. This is mechanized by means of *partial rewriting*, which is a novel rewriting technique that is obtained by replacing the traditional pattern-matching of term rewriting with a new mechanism based on *homeomorphic embedding* (cf. [15]).

3.3. Partial rewriting

Partial rewriting extracts "some pieces of information" from a document, pieces them together, and then rewrites the glued term. The assembling is done by means of a single embedding relation, which allow us to verify whether a given XML document template is somehow "enclosed" within another one and thus replaces the traditional pattern matching mechanism [30].

We give a definition of partial embedding, $\trianglelefteq$, which is an adaptation of the homeomorphic embedding proposed in [31], where: (i) a simpler treatment of the variables is considered; (ii) function symbols with variable arities are allowed; (iii) the relative positions of the arguments of terms are ignored (i.e., $f(a, b)$ is not distinguishable from $f(b, a)$); and (iv) we ignore the usual *diving* rule[4] [31].

Definition 3.3 (partial embedding) *The* partial embedding *relation*

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

on XML document templates is the least relation that satisfies:
1. $X \trianglelefteq t$, *for all* $X \in \mathcal{V}$ *and* $t \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$.
2. $f(t_1, \ldots, t_m) \trianglelefteq g(s_1, \ldots, s_n)$ *iff* $f \equiv g$ *and*
$t_i \trianglelefteq s_{\pi(i)}$, *for* $i = 1, \ldots, m$, *and injective function*
$\pi : \{1, \ldots, m\} \to \{1, \ldots, n\}$.

Whenever $s \trianglelefteq t$, we say that t *embeds* s (or s *is embedded* or "*recognized*" in t). The intuition behind the above definition is that the structure of the template s appears within the template t

Now we are ready to introduce the partial rewrite relation between XML document templates. In order to keep our description simple, w.l.o.g. for the time being we disregard conditions and/or quantifiers from the XML specification rules. Roughly speaking, given an XML specification rule $\lambda \rightharpoonup \rho$, partial rewriting allows us to extract from a given XML document s a subpart of s that is embedded in a ground instance of λ, and

[4] The diving rule allows one to "strike out" a part of the term on the right-hand side of the relation $\trianglelefteq$. Formally, $s \trianglelefteq f(t_1, \ldots\ t_n)$, if $s \trianglelefteq t_i$, for some i.

then replace s by a reduced, ground instance of ρ. Let $s, t \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. Then, s *partially rewrites* to t, in symbols $s \overset{r,u,\sigma}{\rightharpoonup} t$, iff there exist a rule $r = \lambda \rightharpoonup \rho$, a substitution σ, and a position $u \in O_{\mathcal{T}ag}(s)$ such that (i) $\lambda\sigma \trianglelefteq s_{|u}$, and (ii) $t = \mathit{Reduce}(\rho\sigma, R)$, where function $\mathit{Reduce}(m, R)$ computes, by standard term rewriting, the irreducible form of term m in R. Note that the context of the selected reducible expression $s_{|u}$ is disregarded after each partial rewrite step. By notation $s \rightharpoonup_I t$, we denote that s is partially rewritten to t using a rule that belongs to the set I. Also, when the context is clear, we simply write $s \rightharpoonup t$. The transitive (resp., transitive and reflexive) closure of the relation $\rightharpoonup$ is denoted by $\rightharpoonup^+$ (resp., $\rightharpoonup^*$).

3.4. XML verification methodology

Correctness (resp. completeness) errors are detected by applying XML correctness (resp. completeness) specification rules to a given faulty XML repository. We classify the kinds of errors that can be found in an XML repository in terms of the different outputs delivered by our verification technique, when it is fed with an XML specification. In the following sections, we will exploit this classification to develop our repairing methodology.

3.4.1. **Applying the correctness rules to XML documents**

Given an XML specification (R, I_{CN}, I_{CM}), a correctness error is discovered in an XML document p, whenever p is partially rewritten to the constant error using the unconditional part of some correctness rule $\lambda \rightharpoonup \mathit{error} \mid C \in I_{CN}$, and the associated condition C holds in R w.r.t. the computed substitution.

More formally,

Definition 3.4 (correctness error) *Let W be an XML repository, and let (R, I_{CN}, I_{CM}) be an XML specification. Then, the tuple $(p, w, \lambda, \sigma, C)$ is a* correctness error *iff there exist $r_{CN} = \lambda \rightharpoonup \mathit{error} \mid C \in I_{CN}$, a substitution σ, and $w \in O_{\mathcal{T}ag}(p)$, such that $p \overset{r_{CN},w,\sigma}{\rightharpoonup} \mathit{error}$ and C holds in R w.r.t. σ.*

Given a correctness error $(p, w, \lambda, \sigma, C)$, $\lambda\sigma$ represents the erroneous information that is embedded in a subterm of the XML document p, namely $p_{|w}$.

We denote the set of all correctness errors of an XML repository W w.r.t. a set of correctness rules I_{CN} by $E_{CN}(W)$. When no confusion can arise, we just write E_{CN}.

3.4.2. **Applying the completeness rules to XML documents**

Completeness errors are detected by partially rewriting XML documents using completeness rules.

Roughly speaking, given the unquantified part $\lambda \rightharpoonup \rho$ of a completeness rule and an XML document p embedding an instance $\lambda\sigma$ of λ, we partially rewrite p, that is, we replace p that embeds $\lambda\sigma$ with the instantiated right-hand side $\rho\sigma$, in symbols $p \rightharpoonup \rho\sigma$. Each term $\rho\sigma$ generated by partial rewriting is considered to be a *requirement*. Then, from all the requirements computed by partially rewriting the XML documents, new requirements are generated by applying further partial rewriting steps.

Finally, by a new partial embedding test, for each computed requirement, we check whether it is recognized within the XML repository. If the test fails, a completeness error is signaled.

With regard to completeness errors, we can distinguish three classes of errors: (i) *missing XML documents*; (ii) *universal completeness errors*; (iii) *existential completeness errors.*

Definition 3.5 (Missing XML document) *Let W be an XML repository, and let (R, I_{CN}, I_{CM}) be an XML specification. Then the pair $(p \rightharpoonup^+_{I_{CM}} req, W)$ is a* missing XML document *error if there exists $p \in W$ s.t. $p \rightharpoonup^+_{I_{CM}} req$ and $req \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ does not belong to W.*

When a missing XML document error is detected, the pair (req, W) signals that the expression *req* does not appear in the whole XML repository W.

In order to formalize existential as well as universal completeness errors, the following auxiliary definition is helpful.

Definition 3.6 *Let P be a set of terms in $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$, and let $req \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ be a term. We say that P is* universally *(resp.* existentially*)* complete *w.r.t. req iff for each $p \in P$ (resp. for some $p \in P$), there exists $w \in O_{\mathcal{T}ag}(p)$ s.t. $req \trianglelefteq p_{|w}$.*

Definition 3.7 (Universal completeness error) *Let W be an XML repository, $P \subseteq W$, and $p \in W$. Let (R, I_{CN}, I_{CM}) be an XML specification and $p \rightharpoonup^+_{I_{CM}} req$. Then, the triple $(p \rightharpoonup^+_{I_{CM}} req, P, A)$ is a* universal completeness error *if P is not universally complete w.r.t. req.*

Definition 3.8 (Existential completeness error) *Let W be an XML repository, $P \subseteq W$, and $p \in W$. Let (R, I_{CN}, I_{CM}) be an XML specification and $p \rightharpoonup^+_{I_{CM}} req$. Then, the triple $(p \rightharpoonup^+_{I_{CM}} req, P, E)$ is an* existential completeness error*, if P is not existentially complete w.r.t. req.*

The requirement *req* appearing in a completeness error (i.e. missing XML document, existential/universal completeness error) is called *missing* requirement.

Note that Definition 3.7 (resp. Definition 3.8) formalizes the fact that the XML repository W fails to fullfil the property that a piece of information must occur in *all* (resp. *some*) XML documents of a given subset P of W. We denote by $E_{CM}(W)$ the set that contains all the completeness errors w.r.t. I_{CM} for an XML repository W. When no confusion can arise, we just write E_{CM}.

Given a completeness error e, $RSeq(e)$ denotes the partial rewriting sequence included in e, and $Req(e)$ specifies the missing requirement included in e. In other words, if $e = (p \rightharpoonup^+ req, W)$ or $e = (p \rightharpoonup^+ req, P, E)$, or $e = (p \rightharpoonup^+ req, P, A)$, $RSeq(e) = p \rightharpoonup^+ req$ and $Req(e) = req$, respectively.

4. Repairing Correctness and Completeness errors

In order to repair a faulty XML repository, we introduce four repair actions that will be used as primitives within the repair strategies. The primitive repair actions that we consider are the following:

- **change**(p, w, t), which replaces the subterm $p|_w$ in p with the term t;
- **insert**(p, w, t), which modifies the term p by adding the term t into $p_{|w}$;
- **add**(p, W), which adds the XML document p to the XML repository W;
- **delete**(p, t), which deletes all the occurrences of the term t in the XML document p.

Each repair action returns the transformed XML document/repository after its execution. Note that it is possible for a particular error to be repaired by means of different actions. For instance, a correctness error can be fixed by either deleting the incorrect/forbidden information, or by changing the data that give rise to that error. Similarly, a completeness error can be fixed by either (i) inserting the missing information, or (ii) deleting all the data in the XML repository that caused that error. Moreover, the modification or insertion of arbitrary information may cause the appearance of new correctness errors. In order to avoid this, we have to ensure that the data considered for insertion are correct w.r.t. the XML specification, i.e., they cannot fire any correctness rule. The following definition formalizes this idea.

Definition 4.1 *Let (R, I_{CN}, I_{CM}) be an XML specification, and let $p \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ be an XML document. Then, p is* correct *w.r.t. (I_{CN}, R), iff for each $w \in O_{\mathcal{T}ag}(p)$ and $(\lambda \rightharpoonup \rho \mid C) \in I_{CN}$, either (i) there is no σ s.t. $\lambda\sigma \trianglelefteq p_{|w}$; or (ii) $\lambda\sigma \trianglelefteq p_{|w}$, but C does not hold in R w.r.t. σ.*

In the following, we develop some basic approaches that get rid of both, correctness and completeness errors from an XML repository. We proceed as follows. First, we deal with correctness errors. We define two repair techniques that allow wrong information to be removed from an XML repository: *correctness through deletion*, and *correctness through change*. Then, we focus our attention on completeness errors and define two repair techniques that fix completeness errors: *completeness through insertion*, and *completeness through deletion*.

4.1. Fixing Correctness Errors

Throughout this section, we will consider a given XML repository W, an XML specification (R, I_{CN}, I_{CM}) and the set $E_{CN}(W) \neq \emptyset$ of the correctness errors w.r.t. (I_{CN}, R) for W.

Given $e = (p, w, \lambda, \sigma, C) \in E_{CN}$, e can be repaired in two distinct ways: we can decide either (i) to remove the wrong content $\lambda\sigma$ from the XML document p (specifically, from $p_{|w}$), or (ii) to change $\lambda\sigma$ into a piece of correct information.

4.1.1. Correctness through Deletion

In this case, we simply remove all the occurrences of the subterm $p_{|w}$ of the XML document p that contain the wrong information $\lambda\sigma$ by applying the repair action $\mathbf{delete}(p, p_{|w})$.[5] This way, we delete those data, that fired the correctness rule responsible for the error, from the faulty page p.

Example 4.2

Consider the XML repository in Figure 1 and the XML specification in Example 3.2. The rule r_5 of the specification generates the following correctness error

$$\begin{aligned}&(\texttt{P4}, 3, \texttt{email(X)}, \{\texttt{X/gialli@gmail.com}\},\\ &\ \texttt{inRegExp(X, [:Text:]}^{+}\texttt{@myresearchgroup.org}) = \texttt{false})\end{aligned}$$

since the email address in page P4 is not valid according to the specification encoded in rule r_5. The error can be fixed by applying the repair action $\mathbf{delete}(\texttt{P4}, \texttt{P4}_{|3})$, that removes the wrong email from page P4.

4.1.2. Correctness through Change

Given a correctness error $e = (p, w, \lambda, \sigma, C) \in E_{CN}$, we replace the subterm $p_{|w}$ of the XML document p with a new term t that is introduced by the user. The new term t must fulfill some conditions that are automatically provided and checked in order to guarantee the correctness of the inserted information. In the following, we show how these constraints can be computed.

Roughly speaking, whenever we fix some wrong data by executing a repair action $\mathbf{change}(p, w, t)$, it is not enough to ensure that the term t being introduced has no correctness errors (i.e., it is correct w.r.t. (I_{CN}, R)); we also need to consider t within the context that surrounds it in p. If we do not pay attention to such a global condition, some subtle correctness errors might arise as witnessed by the following example.

Example 4.3

Consider the XML document $\texttt{p} = \texttt{f(g(a), b, h(c))}$ and the following correctness rule set

$$I_{CN} = \{(\texttt{r1})\ \texttt{f(g(b))} \rightharpoonup \texttt{error}, \quad (\texttt{r2})\ \texttt{g(a)} \rightharpoonup \texttt{error}\}.$$

The XML document p contains a correctness error according to rule (r2). By executing the repair action

$$\mathbf{change}(\texttt{f(g(a), b, h(c))}, 1, \texttt{g(b)}).$$

[5] Note that, instead of removing the whole subterm $p_{|w}$, it would also be possible to provide a more precise though time-expensive implementation of the **delete** action that only gets rid of the part $\lambda\sigma$ of $p_{|w}$ that is responsible for the correctness error.

the XML document $\mathtt{f(g(b),b,h(c))}$ is obtained from p. Although the term $\mathtt{g(b)}$ is correct w.r.t. (I_{CN}, R)) (i.e., it does not introduce any new correctness error), the replacement of $\mathtt{g(a)}$ with $\mathtt{g(b)}$ in p produces a new correctness error that is recognizable by rule ($\mathtt{r1}$).

In order to avoid these kinds of undesirable repairs, we define the following global correctness property, which simply prevents a newly introduced term t from firing any correctness rule when inserted in the XML document to be fixed.

The following definition is auxiliary. Let $s, t \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$ s.t. $s \trianglelefteq t$. We define the set $Emb_s(t)$ as the set of all the positions in t that embed some subterm of s. For instance, consider the terms $f(k, g(c))$, and $f(b, g(c), k)$. Then, $f(k, g(c)) \trianglelefteq f(b, g(c), k)$, and

$$Emb_{f(k,g(c))}(f(b, g(c), k)) = \{\Lambda, 2, 2.1, 3\}$$

Definition 4.4 *Let (R, I_{CN}, I_{CM}) be an XML specification, $p' =$ **change**(p, w, t) be a repair action that produces the XML document p'. Then,* **change**(p, w, t) obeys the global correctness property *if, for each correctness error $e = (p', w', \lambda, \sigma, C)$ such that $w' \leq w$, satisfies that*

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

The idea behind Definition 4.4 is that any error e in the new page $p' =$ **change**(p, w, t), which is obtained by inserting term t within p, is not a consequence of this change but already existed in a different subterm of p. For this purpose, we require that (the set of positions of) the wrong information $\lambda\sigma$ does not "overlap" the considered term t.

Example 4.5

Consider Example 4.3 again. The repair action

$$\textbf{change}(\mathtt{f(g(a),b,h(c))}, 1, \mathtt{g(b)})$$

does not obey the global correctness property. Indeed, it generates an XML document $\mathtt{f(g(b),b,h(c))}$ that contains a new correctness error w.r.t. the considered specification.

The following proposition holds.

Proposition 4.6 *Let (R, I_{CN}, I_{CM}) be an XML specification, and let W be an XML repository. Let $E_{CN}(W)$ be the set of correctness errors that appear in W, and let $(p, w, \lambda, \sigma, C) \in E_{CN}(W)$ be a correctness error. By executing a repair action* **delete**$(p, p_{|w})$ *(resp.,* **change**(p, w, t) *such that it obeys the global correctness property and t is correct w.r.t. (I_{CN}, R)), we have that*

$$|E_{CN}(W')| < |E_{CN}(W)|$$

where

$$W' = (W \setminus \{p\}) \cup \{\mathbf{delete}(p, p_{|w})\}$$
$$(\textit{resp.}, \; W' = (W \setminus \{p\}) \cup \{\mathbf{change}(p, w, t)\})$$

Proof. We prove the two cases separately.

Case (i). Assume that the repair action $\mathbf{delete}(p, p_w)$ is executed. In this case, the proof is immediate, since the correctness error $(p, w, \lambda, \sigma, C)$ is repaired by executing $\mathbf{delete}(p, p_{|w})$, and no new information is added to the XML repository, hence, no extra correctness errors can be introduced.

Case (ii). Assume that the repair action $\mathbf{change}(p, w, t)$ is executed. By hypothesis, $\mathbf{change}(p, w, t)$ obeys the global correctness property and t is correct w.r.t. (I_{CN}, R). The proof for this case is also immediate. It suffices to observe that t is free of correctness errors since it is correct w.r.t. (I_{CN}, R). Furthermore, Definition 4.4 (global correctness property) prevents new errors from being introduced by the application of the repair action $\mathbf{change}(p, w, t)$. Hence, the execution of $\mathbf{change}(p, w, t)$ repairs the error $(p, w, \lambda, \sigma, C)$. Therefore, $|W'| = |(W \setminus \{p\}) \cup \{\mathbf{change}(p, w, t)\}| < |W|$.

■

Proposition 4.6 states that the application of *Correctness through Deletion* and *Correctness through Change* techniques decreases the number of correctness errors of the original XML repository at least by one. Therefore, the correctness of an XML repository can be achieved by repeated applications of *Correctness through Deletion* and *Correctness through Change* techniques to distinct correctness errors. Also, note that each application of the considered repair techniques involves the execution of a **delete** or a **change** action that might repair multiple correctness errors, simultaneously — e.g., given two correctness errors e_1 and e_2 in the same subterm t of an XML document p, by deleting t we repair both e_1 and e_2 in p. This fact can be exploited to develop optimized, repair strategies that free an XML repository from correctness errors (See Section 5 for further details).

4.2. Fixing Completeness Errors

In this section, we address the problem of fixing completeness errors through suitable repair actions. Without loss of generality, we assume that W is an incomplete but correct XML repository w.r.t. a given XML specification (R, I_{CN}, I_{CM}). Such an assumption will allow us to design repairs that "complete" the XML repository and do not introduce any incorrect information.

Let $E_{CM}(W)$ be the set of completeness errors w.r.t. I_{CM} for the XML repository W. Any completeness error that belongs to $E_{CM}(W)$ can be repaired following distinct techniques and, thus, by applying distinct repair actions. On the one hand, we can think of adding the needed data whenever an XML document or a piece of information in an XML document is missing. On the other hand, all the information that caused the error might be removed to get rid of the bug. In both cases, we make use of *compound* repair actions, that is, sequences of repair actions that must be executed to fix a given completeness error.

In the following, we distinguish and discuss about the two possible repair techniques mentioned above.

4.2.1. Completeness through Insertion

We consider two distinct kinds of repair actions, namely $\mathbf{add}(p, W)$ and $\mathbf{insert}(p, w, t)$, according to the kind of completeness error we have to fix. The former action adds a new XML document p to an XML repository W and thus will be employed whenever a missing XML document error needs to be fixed. The latter action allows us to add a new piece of information t to (a subterm of) an incomplete XML document p, and therefore is suitable to repair universal as well as existential completeness errors.

Both the **add** action and the **insert** action introduce new information in the XML repository that might be potentially dangerous since it may contain erroneous as well as incomplete data. It is therefore important to constrain the kind of information a user may want to add. In order to preserve correctness, we compel the user to only insert correct information in the sense of Definition 4.1. Hence, the new data to be added by the execution of some repair action cannot subsequently fire a correctness rule.

Additionally, we want to prevent the execution of the repair actions from introducing new completeness errors, that is, we just want to fix all and only the initial set of completeness errors of the XML repository W, namely $E_{CM}(W)$.

Definition 4.7 (acceptable insertion/addition w.r.t. completeness) *Let* (R, I_{CN}, I_{CM}) *be an XML specification, and let* W *be an XML repository. Let* $E_{CM}(W)$ *be the set of completeness errors of* W*.*

- *the repair action* $p_1 = \mathbf{insert}(p, w, t)$ *is* acceptable *w.r.t.* (R, I_{CN}, I_{CM}) *and* W *iff*
  1. p_1 *is correct w.r.t.* (I_{CN}, R)*;*
  2. $Req(e) \trianglelefteq t_{|w}$, $w \in O_{\mathcal{T}ag}(t)$*, for some* $e \in E_{CM}(W)$*;*
  3. *if* $W' = (W \setminus \{p\}) \cup \{p_1\}$*, then* $E_{CM}(W') \subseteq E_{CM}(W)$*.*
- *the repair action* $\mathbf{add}(p_2, W)$ *is* acceptable *w.r.t.* (R, I_{CN}, I_{CM}) *and* W *iff*
  1. p_2 *is correct w.r.t.* (I_{CN}, R)*;*
  2. $Req(e) \trianglelefteq p_{2|w}$, $w \in O_{\mathcal{T}ag}(p_2)$*, for some* $e \in E_{CM}(W)$*;*
  3. *if* $W' = \mathbf{add}(p_2, W)$*, then* $E_{CM}(W') \subseteq E_{CM}(W)$*.*

Definition 4.7 guarantees that the information that is added by **insert** and **add** actions is not only correct but it does not yield any new completeness error either. More precisely, the number of completeness errors does not increase by the effect of the execution of these repair actions.

Example 4.8

Consider the XML specification of Example 3.2 and the universal completeness error, detected by rule $\mathtt{r}_2$, $(\mathtt{P2} \rightharpoonup \mathtt{person(status(professor), teaching)}, \{\mathtt{P2}, \mathtt{P4}\}, \mathtt{A})$ where P2 and P4 are the personal records of Mario Rossi and Anna Gialli in the XML repository of Figure 1. To fix the error, we should add some teaching information to the XML document P2, while XML document P4 is already complete w.r.t. the requirement $\mathtt{person(status(professor), teaching)}$. Consider the pieces of information

$$t_1 = \mathtt{teaching(course(title(Logic), syllabus(In\ this\ course\ we\ present} \ldots)))$$
$$t_2 = \mathtt{teaching(courselink(url(www.mycourse.com), urlname(LogicCourse)))}$$

If we introduce term t_1, the corresponding **insert** action is acceptable. However, the insertion of term t_2 would produce a new completeness error (namely, a broken link error), since rule $\mathtt{r}_4$ requires an XML document, that describes a course, for every course link that occurs in the repository.

The *Completeness through Insertion* technique makes use of the compound, repair action **RepairByInsert** of Definition 4.9 that may involve the execution of several **insert** actions.

Definition 4.9 (RepairByInsert) *Let W be an XML repository, and (R, I_{CN}, I_{CM}) be an XML specification. Let $E_{CM}(W)$ be the set of completeness errors w.r.t. I_{CM} of W and $e \in E_{CM}(W)$.*

The compound, repair action **RepairByInsert**(W, e) *is defined according to the kind of the considered completeness error e.*

Missing XML document errors. *If $e = (p \rightharpoonup^{+}_{I_{CM}} req, W)$,*
RepairByInsert(W, e) *transforms W into W' as follows:*

$$W' = \mathbf{add}(p', W)$$

where $p' \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$, $req \trianglelefteq p'_{|w}$ for some $w \in O_{\mathcal{T}ag}(p)$, and $\mathbf{add}(p', W)$ is acceptable w.r.t. (R, I_{CN}, I_{CM}) and W.

Existential completeness errors. *If $e = (p \rightharpoonup^{+}_{I_{CM}} req, \{p_1, p_2, \ldots, p_n\}, E)$,*
RepairByInsert(W, e) *transforms W into W' as follows:*

$$W' = (W \setminus \{p_i\}) \cup \{\mathbf{insert}(p_i, w, t)\}$$

where $req \trianglelefteq t$ for some $t \in \tau(\mathcal{T}ext \cup \mathcal{T}ag)$, $p_i \in W$ for some $i = 1, \ldots, n$, $w \in O_{\mathcal{T}ag}(p_i)$, and $\mathbf{insert}(p_i, w, t)$ *is acceptable w.r.t. (R, I_{CN}, I_{CM}) and W.*

Universal completeness errors. *If* $e = (p \rightharpoonup^{+}_{I_{CM}} req, \{p_1, p_2, \ldots, p_n\}, A)$, **RepairByInsert**$(W, e)$ *transforms* W *into* W' *as follows.*

Let $P' = \{p' \in \{p_1, \ldots, p_n\} \mid req \not\trianglelefteq p'_{|w'}, \forall\, w' \in O_{\mathcal{T}ag}(p')\}$

$$\begin{aligned} W' = (W \setminus P') \cup \{\textbf{insert}(p', w', t') \mid & p' \in P', \\ & \exists t' \in \tau(\mathcal{T}ext \cup \mathcal{T}ag), w_i \in O_{\mathcal{T}ag}(p') \text{ s.t.} \\ & req \trianglelefteq t', \\ & \textbf{insert}(p', w', t') \text{ is acceptable w.r.t.} \\ & (R, I_{CN}, I_{CM}) \text{ and } W\} \end{aligned}$$

Note that the new pieces of information, that are inserted by **RepairByInsert** via **add** and **insert** actions, are typically provided by the expert. In the case when an **insert**(p, w, t) action is executed, the expert also specifies the position w in p to which attach the chosen term t.

As an immediate consequence of Definition 4.9, we have the following proposition.

Proposition 4.10 *Let* (R, I_{CN}, I_{CM}) *be an XML specification, and let* W *be an XML repository. Let* $E_{CM}(W)$ *be the set of completeness errors of* W *w.r.t.* I_{CM}*, and* $e \in E_{CM}(W)$*. Then,* **RepairByInsert**(W, e) *transforms* W *into an XML repository* W' *such that* $|E_{CM}(W')| < |E_{CM}(W)|$.

Proof. The proof directly follows by Definition 4.9. We distinguish the following three cases according to the kind of error e that is considered.

$\mathbf{e} = (\mathbf{p} \rightharpoonup^{+}_{\mathbf{I_{CM}}} req, \mathbf{W})$. By Definition 4.9, **RepairByInsert**(W, e) adds an XML document p', that embeds *req*, to W through the repair action **add**(p', W), generating a new XML repository $W' = W \cup \{p'\}$ in which e is repaired. Since **add**(p', W) is acceptable w.r.t. (R, I_{CN}, I_{CM}) and W, we have that $E_{CM}(W') \subseteq E_{CM}(W)$. Furthermore, since **add**(p', W) repairs the completeness error e, $E_{CM}(W') \subset E_{CM}(W)$, and hence $|E_{CM}(W')| < |E_{CM}(W)|$.

$\mathbf{e} = (\mathbf{p} \rightharpoonup^{+}_{\mathbf{I_{CM}}} req, \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}, \mathbf{E})$**.** By Definition 4.9, **RepairByInsert**(W, e) adds a term t, that embeds *req*, to an XML document $p_i \in \{p_1, p_2, \ldots, p_n\} \subseteq W$ through the repair action **insert**(p_i, w, t), generating a new XML repository $W' = (W \setminus \{p\}) \cup \{p'\}$ in which e is repaired. Since **insert**(p_i, w, t) is acceptable w.r.t. (R, I_{CN}, I_{CM}) and W, we have that $E_{CM}(W') \subseteq E_{CM}(W)$. Furthermore, since **insert**(p_i, w, t) repairs the completeness error e, $E_{CM}(W') \subset E_{CM}(W)$, and hence $|E_{CM}(W')| < |E_{CM}(W)|$.

$\mathbf{e} = (\mathbf{p} \rightharpoonup^{+}_{\mathbf{I_{CM}}} req, \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}, \mathbf{A})$**.** By Definition 4.9, **RepairByInsert**(W, e) executes a sequence of **insert**(p_i, w_i, t_i) actions that adds a term t_i, that embeds *req*, to all the XML documents $p_i \in \{p_1, p_2, \ldots, p_n\} \subseteq W$ such that

$$req \not\trianglelefteq p_{i|w_i}, \text{ for all } w_i \in O_{\mathcal{T}ag}(p_i).$$

This generates a new XML repository W' in which e is repaired. Since each $\mathbf{insert}(p_i, w_i, t_i)$ is acceptable w.r.t. (R, I_{CN}, I_{CM}) and W, we have that $E_{CM}(W') \subseteq E_{CM}(W)$. Furthermore, since the sequence of $\mathbf{insert}(p_i, w_i, t_i)$ actions repairs the completeness error e, $E_{CM}(W') \subset E_{CM}(W)$, and hence $|E_{CM}(W')| < |E_{CM}(W)|$.

■

4.2.2. Completeness through Deletion

When dealing with completeness errors, sometimes it is more convenient to delete incomplete data instead of completing them. In particular, we can identify two scenarios where this option can be very useful: 1) when there is no expert who can guide the completeness through insertion strategy, which may happen if the XML content is automatically generated by a dynamic application which does not respect the XML specification, or 2) whenever we want to get rid of out-of-date information as illustrated in Example 4.13 below. The main idea of *Completeness through Deletion* is to remove all the information in the XML repository that caused a given completeness error. This approach is independent of the kind of completeness error we are handling, since the missing requirement is computed in the same way for all of the three kinds of errors by partially rewriting the original XML documents of the XML repository. In other words, given the missing requirement *req* of a completeness error e (i.e., a missing page error $e = (p \rightharpoonup^+ req, W)$, or an existential completeness error $e = (p \rightharpoonup^+ req, P, E)$, or a universal completeness error $e = (p \rightharpoonup^+ req, P, A)$), there exists an XML document $p \in W$ such that $p \rightharpoonup^+ req$.

Therefore, we proceed by computing and eliminating from the XML repository all the occurrences of the term that started the partial rewrite sequence that leads to *req*. As in the case of the insertion, we have to be careful about the effects of the execution of the repair actions. More precisely, we do not want the execution of any **delete** action to introduce new completeness errors[6]. For this purpose, we consider the following notion of *acceptable* delete action.

Definition 4.11 (acceptable deletion w.r.t. completeness) *Let (R, I_{CN}, I_{CM}) be an XML specification, and let W be an XML repository. Let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. The repair action $p_1 = \mathbf{delete}(p, t)$ is* acceptable *w.r.t. (R, I_{CN}, I_{CM}) and W iff $E_{CM}(W') \subseteq E_{CM}(W)$, where $W' = (W \setminus \{p\}) \cup \{p_1\}$.*

The *Completeness through Deletion* technique exploits the **RepairByDelete** compound, repair action of Definition 4.12, which removes all the information responsible for the completeness error by possibly executing multiple **delete** actions.

[6] Correctness errors are not considered, since the execution of a **delete** action cannot generate any new correctness error.

Definition 4.12 (RepairByDelete) *Let (R, I_{CN}, I_{CM}) be an XML specification, and let W be an XML repository. Let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}, and $e \in E_{CM}(W)$. Then,* **RepairByDelete**(W, e) *transforms W into the XML repository W' in the following way.*

For each $p \overset{r,w,\sigma}{\rightharpoonup} t \rightharpoonup^* Req(e)$ where $p \in W$,

$$\begin{aligned} W' = &\{p' \in W \mid p_{|w} \ntrianglelefteq p'_{|w'}, \forall\, w' \in O_{\mathcal{T}ag}(p')\} \cup \\ &\quad \{\mathbf{delete}(p', p_{|w}) \mid \mathbf{delete}(p', p_{|w}) \textit{ is acceptable w.r.t. } (R, I_{CN}, I_{CM}) \textit{ and } W, \\ &\qquad\qquad p' \in W, p_{|w} \trianglelefteq p'_{|w'}, \textit{ for some } w' \in O_{\mathcal{T}ag}(p')\} \end{aligned}$$

Example 4.13

Consider the XML specification of Example 3.2, the XML repository W of Figure 1, and the missing XML document error

$$e_{\texttt{P1}} = (\texttt{P1} \rightharpoonup \texttt{person}(\texttt{fullname}(\texttt{ugoblu}), \texttt{status}), \texttt{W})$$

that can be detected in W by using the completeness rule r_1. The missing document is obtained by means of the following partial rewrite sequences:

$\texttt{P1} \overset{r_1,4,\sigma_1}{\rightharpoonup} \texttt{person}(\texttt{fullname}(\texttt{ugoblu}), \texttt{status})$

$\texttt{P5} \overset{r_3,1,\sigma_2}{\rightharpoonup} \texttt{member}(\texttt{name}(\texttt{ugo}), \texttt{surname}(\texttt{blu})) \overset{r_1,4,\sigma_3}{\rightharpoonup} \texttt{person}(\texttt{fullname}(\texttt{ugoblu}), \texttt{status})$

By applying **RepairByDelete**$(W, e_{\texttt{P1}})$, we delete all the information regarding the group membership and the publications of `Ugo Blu`, that is, we remove the subterm $\texttt{P1}_{|\texttt{4}}$ from page `P1` and the subterm $\texttt{P5}_{|\texttt{1}}$ from page `P5`. This amounts to erasing all the data regarding `Ugo Blu` from the considered XML repository.

The following result holds.

Proposition 4.14 *Let (R, I_{CN}, I_{CM}) be an XML specification, and let W be an XML repository. Let $E_{CM}(W)$ be the set of completeness errors of W, and $e \in E_{CM}(W)$. By executing the* **RepairByDelete**(W, e) *repair action, W is transformed into an XML repository W' such that $|E_{CM}(W')| < |E_{CM}(W)|$.*

Proof. Immediate by Definition 4.12. ■

Observe that Proposition 4.10 and Proposition 4.14 state that the application of **RepairByInsert** and **RepairByDelete** to an XML repository W decreases the number of completeness errors in W. Therefore, completeness of an XML repository can be achieved by executing the compound, repair actions **RepairByInsert** and **RepairByDelete** on the distinct completeness errors that appear in the considered repository. Completion strategies are treated in detail in Section 6.

5. Achieving Correctness of XML Repositories via Correction Strategies

In this section, we carry out a systematic analysis on the relations among correctness errors that we use to formalize two automated correction strategies: the $\mathbb{NAR}$ strategy allows the number of repair *actions* that are executed to be reduced, while the $\mathbb{NDR}$ strategy reduces the amount of information *data* to be changed/removed in order to fix all the correctness errors of a faulty XML repository.

Roughly speaking, a correction strategy is a finite sequence of **delete** and **change** repair actions whose execution allows a faulty XML document to be fixed. We assume that the considered strategies make use of the basic repairing techniques *Correctness through Delete* and *Correctness through Change* of Section 4.1 that allow us to safely apply **delete** and **change** actions to XML documents, that is, their application to XML documents does not generate any new bug.

5.1. Correctness Error Dependencies

Typically, a given XML document can contain several correctness errors that may be interrelated. Since the execution of a repair action might fix a number of related correctness errors simultaneously, it is crucial to discover the way an error depends on other errors. In the following, we first analyze the dependencies among correctness errors. Then, we exploit this information to develop the $\mathbb{NAR}$ and $\mathbb{NDR}$ correction strategies.

We relate correctness errors by comparing the positions at which they occur in a given XML document w.r.t. the prefix ordering over term positions $\leq$ which we defined in Section 2. Given a correctness error $e = (p, w, \lambda, \sigma, C) \in E_{CN}(\{p\})$, $pos(e)$ denotes the position at which e occur in the XML document p, that is, $pos(e) = w$. We say that two correctness errors e_1 and e_2 are not *comparable* iff $pos(e_1) \not\leq pos(e_2)$ and $pos(e_2) \not\leq pos(e_1)$. A correctness error e is minimal in a set of correctness errors E iff there does not exist $e' \in E$ such that $pos(e') \leq pos(e)$ and $pos(e') \neq pos(e)$.

Proposition 5.1 *Let p be an XML document, and let $e_i = (p, w_i, \lambda_i, \sigma_i, C_i) \in E_{CN}(\{p\})$, $i = 1, \ldots, n$, such that $pos(e_1) \leq pos(e_2) \leq \ldots \leq pos(e_n)$. The following results hold:*

- *If $p' = \mathbf{change}(p, w_1, t)$, then $E_{CN}(\{p'_{|w_1}\}) = \emptyset$.*
- *If $p' = \mathbf{delete}(p, p_{|w_1})$, then $E_{CN}(\{p'_{|w_1}\}) = \emptyset$.*

Proof. (Sketch) The proof of this result relies on the fact that the errors $e_2, \ldots, e_n$ are located into the subterm $p_{|w_1}$, which is changed or deleted by the action under consideration. Note also that the **change** action always inserts a correct term t into p at position w_1, which implies that not new correctness errors are introduced in $p_{|w_1}$ because of the execution of the **change** repair action. ■

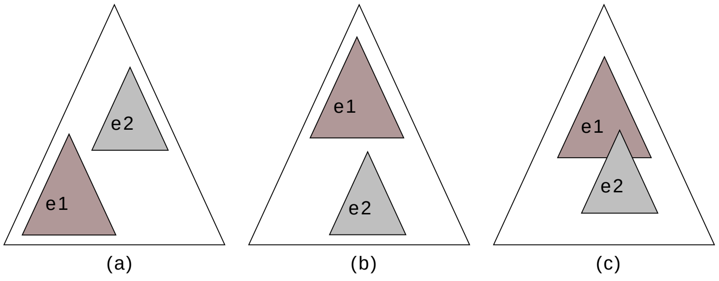


Figure 2: Taxonomy of error dependencies.

Informally, Proposition 5.1 states that repairing a given correctness error $e_1 = (p, w_1, \lambda_1, \sigma_1, C_1)$ allows us to automatically fix any correctness error that is included in the term $p_{|w_1}$.

Let us discuss what happens when errors are not comparable, or we decide to fix an error that is not the smallest in the order. In this case, we are interested in knowing whether it is still possible to fix more than one error at a time. Let us start by providing an auxiliary definition.

Definition 5.2 *Let $e_1 = (p, w_1, \lambda_1, \sigma_1, C_1)$ and $e_2 = (p, w_2, \lambda_2, \sigma_2, C_2)$ be two correctness errors in $E_{CN}(\{p\})$. We say that e_2* overlaps *e_1 at w (in symbols, $e_2 \between_w e_1$), iff (i) $pos(e_1) \leq pos(e_2)$, and (ii) there exists $w = min(w_1.Emb_{\lambda_1}(p_{|w_1}) \cap w_2.Emb_{\lambda_2}(p_{|w_2}))$, where $min(X) = w$ with $w \leq w_i$ for all $w_i \in X$. When position w is not relevant or clear from the context, we simply write e_2* overlaps *e_1 or $e_2 \between e_1$.*

By notation $e_2 \not\between e_1$, we denote that e_2 does not overlap e_1. Given two correctness errors e_1 and e_2 of an XML document p, we can distinguish three possible scenarios:

1. e_1 and e_2 are not comparable w.r.t. $\leq$ (see Figure 2(a));
2. $pos(e_1) \leq pos(e_2)$ and e_2 does not overlap e_1 (see Figure 2(b));
3. e_2 overlaps e_1 (see Figure 2(c)).

In case 1, correctness errors e_1 and e_2 are completely independent, and hence the repair of one of them does not affect the correction of the other one.

Let us now consider case 2. In this case, also by Proposition 5.1, we are able to automatically fix e_2 by just repairing e_1. Note that, fixing e_2 will not help to fix e_1, as stated in the following proposition.

Proposition 5.3 *Let p be an XML document. Let $e_1 = (p, w_1, \lambda_1, \sigma_1, C_1)$ and $e_2 = (p, w_2, \lambda_2, \sigma_2, C_2)$ be two correctness errors in $E_{CN}(\{p\})$ such that $pos(e_1) \leq pos(e_2)$ and $e_2 \not\between e_1$. If $p' = action$, with $action \in \{\mathbf{delete}(p, p_{|w2}), \mathbf{change}(p, w_2, t)\}$, then (i) $E_{CN}(\{p'_{|w_2}\}) = \emptyset$, (ii) $(p', w_1, \lambda_1, \sigma', C_1) \in E_{CN}(\{p'\})$ for some substitution σ'.*

Proof. By contradiction. Assume that by repairing the error e_2, also the error e_1 is repaired. This implies that there exists at least a position v such that $v \in Emb_{\lambda_1}(p)$ and $w_2 \leq v$. Since $w_1 \leq w_2$, then also $w_2 \in Emb_{\lambda_1}(p)$. Thus, $(Emb_{\lambda_1}(p) \cap Emb_{\lambda_2}(p)) \neq \emptyset$, i.e., $e_2 \sqsupset\!\!\sqsubset e_1$, which leads to a contradiction. ■

Example 5.4 ______________________________

Consider the XML document $p = f(g(a), h(b))$ and the following correctness errors

$$e_1 = (p, \Lambda, f(X), \{X/h(b)\}, \emptyset\}),\ e_2 = (p, 2, h(Y), \{Y/b\}, \emptyset)$$

Thus, $pos(e_1) \leq pos(e_2)$, and e_2 does not overlap e_1. Now observe that we can fix e_2 by either removing subterm $h(b)$ or by changing subterm $h(b)$ with a suitable term t. In both cases, such a repair will not fix e_1.

____

In case 3, e_1 and e_2 are not independent since $pos(e_1) \leq pos(e_2)$, and e_2 overlaps e_1. Roughly speaking, the correctness error e_2 is partly "contained" in e_1 and thus fixing e_2 might also yield a fix for e_1. However, this is not always the case as witnessed by the following example.

Example 5.5 ______________________________

Consider the XML document $p = f(g(a), h(b))$ and the following correctness errors $e_1 = (p, \Lambda, f(h(X)), \{X/b\}, \emptyset)$, $e_2 = (p, 2, h(X), \{X/b\}, X = b)$. Thus, $pos(e_1) \leq pos(e_2)$, and e_2 overlaps e_1. We can fix e_2 by changing, for instance, $h(b)$ with $h(a)$. However, such a fix would not automatically repair e_1, while by removing $h(b)$ or by replacing $h(b)$ with one term that does not match any subterm of $f(h(X))$ (e.g. $l(c)$), we would fix both e_2 and e_1 by executing a single repair action.

____

As Example 5.5 illustrates, some conditions are necessary in order to automatically achieve a fix for e_1 by simply correcting e_2. This is formalized in the following proposition.

Proposition 5.6 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let p be an XML document. Let $e_1 = (p, w_1, \lambda_1, \sigma_1, C_1)$ and $e_2 = (p, w_2, \lambda_2, \sigma_2, C_2)$ be two correctness errors in $E_{CN}(\{p\})$ such that $pos(e_1) \leq pos(e_2)$ and e_2 overlaps e_1 at some position w. The following results hold:*

1) *If $p' =$ **delete**$(p, p_{|w_2})$, then*
   (i) $E_{CN}(\{p'_{|w_2}\}) = \emptyset$,
   (ii) $(p', w_1, \lambda_1, \sigma', C_1) \notin E_{CN}(\{p'\})$, for any substitution σ' ;

2) *If $p' =$ **change**(p, w_2, t) and $\lambda_{1|w} \not\trianglelefteq t$, then*
   (i) $E_{CN}(\{p'_{|w_2}\}) = \emptyset$,
   (ii) $(p', w_1, \lambda_1, \sigma', C_1) \notin E_{CN}(\{p'\})$, for any substitution σ';

3) *If* $p' = \mathbf{change}(p, w_2, t)$, $\lambda_{1|w}\sigma' \trianglelefteq t$ *for some substitution* σ', *and* C_1 *does not hold in* R *w.r.t.* σ_1/σ', *then*
   (i) $E_{CN}(\{p'_{|w_2}\}) = \emptyset$,
   (ii) $(p', w_1, \lambda_1, \sigma_1/\sigma', C_1) \notin E_{CN}(\{p'\})$.

Proof. Claim 1 and 2 follow from Proposition 5.1 straightforwardly. The proof of Claim 3 derives from Proposition 5.1 and Proposition 4.6, which establish that no new errors are introduced in the XML document by executing a **change** action. ■

Roughly speaking, Proposition 5.6 states that: (i) whenever a **delete** action is undertaken to fix a correctness error e_2, which overlaps a "smaller" (w.r.t. $\leq$) correctness error e_1, such an action also fixes e_1; (ii) whenever a repair action $p' = \mathbf{change}(p, w_2, t)$ is performed in order to fix e_2, some extra conditions are necessary in order to ensure that the term t to be inserted will automatically fix e_1. Basically, these conditions ensure that either (an instance of) the faulty term λ_1 is not recognized in p' or, if such an instance is identified, the associated condition does not hold.

5.2. Correction Strategies

As we explained in Section 4, a given correctness error e in an XML document p can be fixed by executing a suitable repair **delete** or **change** action a. By (e, a), we denote a *correction pair* that consists of a repair action a that fixes a correctness error e. If (e, a) fixes an error in the XML document p, we say that (e, a) is a correction pair for p. Moreover, by notation $p' = a(p)$ we refer to the execution of the repair action a on the XML document p that returns the XML document p'.

Definition 5.7 (Correction Strategy) *Let* p *be an XML document, and let* $E_{CN}(\{p\})$ *be the set of correctness errors of* p. *A* correction strategy *for* p *is a finite sequence* $\langle(e_1, a_1), \ldots (e_n, a_n)\rangle$ *of correction pairs for* p, *with* $a_1, \ldots a_n$ *repair actions, and* $e_1, \ldots e_n$ *correctness errors in* $E_{CN}(\{p\})$, *such that*

1. $p_0 = p$;
2. $p_i = a_i(p_{i-1})$, $0 < i \leq n$.

and $E_{CN}(\{p_n\}) = \emptyset$.

Roughly speaking, given a faulty XML document p, a correction strategy for p allows all the bugs in p to be fixed by running all the repair actions that occur in the strategy.

The definition of correction strategy for a single XML document can be naturally extended to XML repositories as follows.

Definition 5.8 *Let* $W = \{p_1, \ldots, p_n\}$ *be an XML document. Let* $E_{CN}(W)$ *be the set of correctness errors of* W. *A* correction strategy *for* W *is a sequence* $\langle\mathcal{S}_1, \ldots, \mathcal{S}_n\rangle$, *where* $\mathcal{S}_i$ *is a correction strategy for* p_i, $i = 1, \ldots, n$.

Note that by applying a correction strategy to an XML repository W, we obtain a new repository W' such that $E_{CN}(W') = \emptyset$, that is, the resulting repository W' is free of correctness errors.

As we discussed in Section 5.1, fixing a correctness error may automatically repair other bugs. This fact shows that a correctness strategy does not necessarily contain a correction pair (e, a) for any correctness error e that is found in a faulty XML document. In the following, we describe two semi-automatic correction strategies that exploit the results of Section 5.1.

In both cases, we assume that for any $e \in E_{CN}(\{p\})$, we have a correction pair (e, a) for p at hand that can be executed if this is necessary.

5.2.1. Correction with Action Reduction Strategy

First, let us observe the following fact.

- **Fact 1.** By Proposition 5.1, for every correction pairs $(e, a), (e', a')$ for p, such that $pos(e) \leq pos(e')$, the execution of the repair action a will fix both e and e'. Therefore, fixing a correctness error e, such that e is minimal in $E_{CN}(\{p\})$, will fix all the correctness errors e' in p that are "greater" than e w.r.t. $\leq$, without running any other repair action.

By Fact 1, it is straightforward to see that it suffices to fix those errors that are minimal in $E_{CN}(\{p\})$ to repair the whole XML document p. Let $M(E_{CN}(\{p\}))$ be the set of all correctness errors in $E_{CN}(\{p\})$ that are minimal in $E_{CN}(\{p\})$.

Definition 5.9 (Correction with Action Reduction Strategy, $\mathbb{NAR}$) *Let p be an XML document.*

A Correction with Action Reduction *strategy for p (or* $\mathbb{NAR}(p)$ strategy*) is a sequence $\langle (e_1, a_1), \ldots (e_m, a_m) \rangle$ of correction pairs for p, such that $e_i \in M(E_{CN}(\{p\}))$, for all $i = 1, \ldots m$.*

Given an XML repository $W = \{p_1, \ldots, p_n\}$, a *Correction with Action Reduction* strategy for W (or $\mathbb{NAR}(W)$ *strategy*) is a sequence $\langle \mathcal{T}_1, \ldots, \mathcal{T}_n \rangle$, where $\mathcal{T}_i$ is a $\mathbb{NAR}(p_i)$ strategy, $i = 1, \ldots, n$.

The next proposition states that any given $\mathbb{NAR}(p)$ strategy is a correction strategy for the XML document p, whose number of repair actions is less or equal than the total number of correctness errors detected in p.

Proposition 5.10 *Let p be an XML document. Then, a $\mathbb{NAR}(p)$ strategy is a correction strategy for p such that $|\mathbb{NAR}(p)| \leq |E_{CN}(\{p\})|$.*

Proof. The proof that a $\mathbb{NAR}(p)$ strategy is a correction strategy for p is immediate by the definition of error minimality in $E_{CN}(\{p\})$ and Proposition 5.1. Furthermore, $|\mathbb{NAR}(p)| \leq |E_{CN}(\{p\})|$, because $M(E_{CN}(\{p\})) \subseteq E_{CN}(\{p\})$. ■

Proposition 5.10 can be trivially lifted to XML repositories.

Corollary 5.11 *Let W be an XML repository. Then, a* $\mathbb{NAR}(W)$ strategy *is a correction strategy for W such that* $|\mathbb{NAR}(W)| \leq |E_{CN}(W)|$.

Proof. Immediate by Proposition 5.10 and Definition 5.8. ■

Example 5.12

Consider the XML document $p = f(g(10), h(d),\ 20)$ together with the following correction strategy $\mathcal{T}$ for p that includes two repair actions:

$$\langle ((p, \Lambda, f(g(X), 20), \{X/10\}, \{X < 20\}), \mathbf{change}(p, \Lambda, f(g(20), 15))), \\ ((p, 2, h(Y), \{Y/d\}, \emptyset), \mathbf{delete}(p, h(d)))\rangle$$

The $\mathbb{NAR}(p)$ strategy corresponds to the unary sequence

$$\langle ((p, \Lambda, f(g(X), 20), \{X/10\}, \{X < 20\}), \mathbf{change}(p, \Lambda, f(g(20), 15)))\rangle.$$

Note that $\mathbb{NAR}(p)$ fixes the XML document p by using just one repair action.

5.2.2. Correction with Data Reduction Strategy

The $\mathbb{NAR}$ strategy typically forces the user to modify/introduce a lot of information in an XML document p, even if only minor changes are needed to fix p, as illustrated by the following example.

Example 5.13

Let us consider the XML document $p = f(g(a), k(m(c)), h(a))$ and the set $E_{CN}(\{p\}) = \{(p, \Lambda, f(g(X), h(Y)), \{X/a, Y/a\}, \{X = Y\}), (p, 1, g(a), id, \emptyset)\}$. The $\mathbb{NAR}(p)$ strategy would only fix the minimal error at the root position. This fact might force the user to provide quite a large amount of information if a **change** action is taken, even if a close variant of p would have been enough to fix the bug.

For instance, if the chosen **change** action was $\mathbf{change}(p, \Lambda, f(g(b), k(m(c)), h(a)))$, the user should re-enter the whole XML document p with just a small change at position 1.1.

Instead, if we repaired $(p, 1, g(a), id, \emptyset)$ by means of the repair action $\mathbf{change}(p, 1, g(b))$, the user would correct both errors by introducing a smaller amount of information.

The idea behind the Correction with Data Reduction strategy is thus to "push" the corrections towards the leaves of the XML document as much as possible and to automatically propagate them up to the root position.

Obviously, given two correctness errors e and e' such that $pos(e') \leq pos(e)$, correcting e does not guarantee an automatic repair for e' (see, for instance, Example 5.4). Indeed, by Proposition 5.3, whenever the error e does not overlap a given error e', there is no possibility to automatically lift a correction for e up to e'. On the other hand, under the conditions stated in Proposition 5.6, e' can be repaired by just fixing e, whenever e overlaps e'.

Given an XML document p, we partition $E_{CN}(\{p\})$ into the following two sets:

- $\mathbb{N}\mathrm{OVL}(\{p\}) = \{e \in E_{CN}(\{p\}) \mid \nexists\, e', e' \between e\}$
- $\emptyset\mathrm{VL}(\{p\}) = E_{CN}(\{p\}) \setminus \mathbb{N}\mathrm{OVL}(\{p\})$.

Clearly, $E_{CN}(\{p\}) = \mathbb{N}\mathrm{OVL}(\{p\}) \cup \emptyset\mathrm{VL}(\{p\})$. The correctness errors in $\mathbb{N}\mathrm{OVL}(\{p\})$ (resp., in $\emptyset\mathrm{VL}(\{p\})$) are called *non-overlapping* (resp., *overlapping*) errors. Note that a non-overlapping error e cannot be automatically fixed by executing a repair action on a correctness error e' such that $pos(e) \leq pos(e')$, since correction effects cannot be propagated upwards. However, this is the case of the overlapping errors that may be implicitly affected by other repairs. Actually, the following facts hold.

- **Fact 1.** Given an overlapping error e', there must exist a non-overlapping error e such that $pos(e') \leq pos(e)$.
- **Fact 2.** Let $e, e_0, e_1, \ldots e_n$, $n \geq 0$, be correctness errors. If e is an overlapping error s.t. e_0 overlaps e and $pos(e) \leq pos(e_n) \leq, pos(e_{n-1}) \ldots \leq pos(e_0)$, then e_i overlaps e, $i = 1, \ldots n$.

These facts, together with Proposition 5.1, suggest to us that it suffices to fix only minimal, non-overlapping errors in order to get a repaired XML document. This is because: (i) all the correctness errors that are "greater" (w.r.t. $\leq$) than the considered non-overlapping error will be repaired, as stated by Proposition 5.1; (ii) for each overlapping error e, there is always $e' \in \mathbb{N}\mathrm{OVL}(\{p\})$ such that e' overlaps e; hence by repairing e' we also fix e.

Let $M(\mathbb{N}\mathrm{OVL}(\{p\}))$ be the set of all correctness errors in $\mathbb{N}\mathrm{OVL}(\{p\})$ that are minimal in $\mathbb{N}\mathrm{OVL}(\{p\})$.

The Correction with Data Reduction strategy is formalized as follows.

Definition 5.14 (Correction with Data Reduction Strategy, $\mathbb{NDR}$) *Let p be an XML document. Let $\mathbb{N}OVL(\{p\})$ be the set of non-overlapping correctness errors of p. A* correction with data reduction *strategy for p (or $\mathbb{NDR}(p)$ strategy) is a sequence $\langle(e_1, a_1), \ldots (e_m, a_m)\rangle$, such that (e_i, a_i) is a correction pair for p, $e_i \in M(\mathbb{N}OVL(\{p\})$, for all $i = 1, \ldots m$.*

Given an XML repository $W = \{p_1, \ldots, p_n\}$, a *Correction with Data Reduction Strategy* for W (or $\mathbb{NDR}(W)$ *strategy*) is a sequence $\langle \mathcal{T}_1, \ldots, \mathcal{T}_n\rangle$, where $\mathcal{T}_i$ is a $\mathbb{NDR}(p_i)$ strategy, $i = 1, \ldots, n$.

A $\mathbb{NDR}(p)$ strategy is a correction strategy for an XML document p, whose number of repair actions is less or equal than the total number of correctness errors in p. More formally,

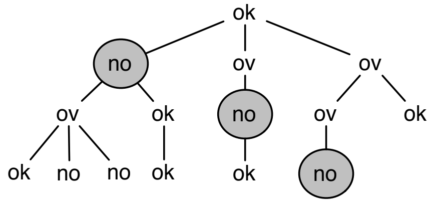


Figure 3: The $\mathbb{NDR}$ strategy

Proposition 5.15 *Let p be an XML document. Then a $\mathbb{NDR}(p)$ strategy is a correction strategy for p such that $|\mathbb{NDR}(p)| \leq |E_{CN}(\{p\})|$.*

Proof. The proof that a $\mathbb{NDR}(p)$ strategy is a correction strategy for p is immediate by the definition of error minimality in $\mathbb{N}$OVL$(\{p\})$ and Proposition 5.1. Furthermore, $|\mathbb{NDR}(p)| \leq |E_{CN}(\{p\})|$, because $M(\mathbb{N}\text{OVL}(\{p\})) \subseteq E_{CN}(\{p\})$. ■

Proposition 5.15 can be trivially lifted to XML repositories.

Corollary 5.16 *Let W be an XML repository. Then, a* $\mathbb{NDR}$ strategy *for W is a correction strategy for W such that $|\mathbb{NDR}(W)| \leq |E_{CN}(W)|$.*

Proof. Immediate by Proposition 5.15 and Definition 5.8. ■

In Figure 3, we show how the *correction with data reduction* strategy works. For the sake of simplicity, we just label each node of the given XML document with: `ok`, if no correctness error is rooted at the considered node; `ov`, if an overlapping error is rooted at the considered node; or `no`, if a non-overlapping error is rooted at the considered node. The XML document contains nine errors, but we just need to fix three errors in order to end up with a repaired XML document. Specifically, these errors correspond to the minimal non-overlapping errors that occur within the XML document.

Example 5.17

Consider the XML document
$p = f(g_1(h_1(a_1, a_2), h_2(b_1, b_2)), g_2(h_3(c_1), h_4))$ and

$$\begin{aligned} E_{CN}(\{p\}) = \{&(p, 1, g_1, id, \emptyset), (p, 1.1, h_1(a_2), id, \emptyset), \\ &(p, 1.1.2, a_2, id, \emptyset), (p, 2.1.1, c_1, id, \emptyset), \\ &(p, 2.1, h_3(c_1), id, \emptyset), \\ &(p, 2, g_2(h_3(X)), \{X/c_1\}, \emptyset)\} \end{aligned}$$

Hence,

$$\begin{aligned} \mathbb{N}\text{OVL}(p) = \{&(p, 1, g_1, id, \emptyset), (p, 1.1.2, a_2, id, \emptyset), \\ &(p, 2.1.1, c_1, id, \emptyset)\} \end{aligned}$$

$$\begin{aligned} \emptyset\text{VL}(p) = &(p, 1.1, h_1(a_2), id, \emptyset), \\ &(p, 2, g_2(h_3(X)), \{X/c_1\}, \emptyset), \\ &(p, 2.1, h_3(c_1), id, \emptyset)\} \end{aligned}$$

A *correction with data reduction* strategy only corrects minimal non-overlapping errors. In this case, a possible $\mathbb{NDR}(p)$ strategy is the sequence:

$$\begin{array}{l}\langle\ ((p,1,g_1,id,\emptyset),\mathbf{delete}(p,1,g_1(h_1(a_1,a_2),h_2(b_1,b_2)),\\ \quad ((p,2.1.1,c_1,id,\emptyset),\mathbf{change}(p,2.1.1,c_4))\ \rangle\end{array}$$

Therefore, the execution of the correction strategy yields the following repaired XML document $f(g_2(h_3(c_4),h_4))$.

Finally, observe that we needed to fix only two errors out of six, and just minor fixes were necessary to make the original XML document correct.

6. Achieving Completeness of XML Repositories via Completion Strategies

The application of a repair action can fix more than one completeness error. This implies that we do not need to execute a different repair action for each detected error, but rather we can choose suitable subsets of errors to act upon.

In this section, we analyze the dependencies among completeness errors in an XML repository. More precisely, we define two partial orderings ($\sqsubseteq_{inf}$ and $\sqsubseteq^{sup}$) that allow completeness errors to be compared by using the embedding relation $\trianglelefteq$. This information is the basis for developing two completion strategies that aim to reduce the number of repair actions to be executed in order to achieve completeness.

By notation $W' = c(W,E)$, we specify the execution of the compound, repair action c on the XML repository W, which returns the XML repository W' where the errors in E have been fixed. By abuse of notation, when E is a singleton $\{e\}$, we simply write $c(W,e)$ instead of $c(W,\{e\})$.

Definition 6.1 (Completion Strategy) *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. A* completion strategy *for W is a finite sequence of compound repair actions $\langle c_0(W_0,E_0),\dots c_n(W_n,E_n)\rangle$, with $E_0,\dots E_n \subseteq E_{CM}(W)$, such that*

1. *$W_0 = W$;*
2. *$W_{i+1} = c_i(W_i,E_i)$ for all $i = 0\dots n$;*

and $E_{CM}(W_{n+1}) = \emptyset$.

Roughly speaking, a completion strategy is a sequence of repair actions that, once executed, allows all the completeness errors in a given XML repository to be repaired.

6.1. Completion with Deletion Reduction Strategy

The Completion with Deletion Reduction strategy is based on the *completeness through deletion* technique of Section 4.2.2 that allows a given completeness errors to be fixed by applying the compound, repair action **RepairByDelete** (see Def. 4.12) which removes all the data responsible for the incompleteness by means of a suitable number of primitive **delete** actions.

To specify the Completion with Deletion Reduction strategy, we first define the following auxiliary preorder $\preceq_{inf}$ on $E_{CM}(W)$.

Definition 6.2 (**$\preceq_{inf}$**) *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $e_1, e_2 \in E_{CM}(W)$ be two completeness errors with $RSeq(e_1) = p_1 \overset{r_1,w_1,\sigma_1}{\rightharpoonup} t_1 \rightharpoonup^* req_1$ and $RSeq(e_2) = p_2 \overset{r_2,w_2,\sigma_2}{\rightharpoonup} t_2 \rightharpoonup^* req_2$. Then,*

$$e_1 \preceq_{inf} e_2 \quad \textit{iff} \quad \textit{there exists } w \in O_{\mathcal{T}ag}(p_{2|w_2}) \textit{ s.t. } p_{1|w_1} \trianglelefteq (p_{2|w_2})_{|w}$$

Roughly speaking, the relation $e_1 \preceq_{inf} e_2$ considers the subterms of p_1 and p_2 that are reduced in the first partial rewrite step of the partial rewrite sequences $RSeq(e_1)$ and $RSeq(e_2)$, namely $p_{1|w_1}$ and $p_{2|w_2}$. Whenever $p_{1|w_1}$ is embedded into a subterm of $p_{2|w_2}$, e_1 is smaller than e_2 w.r.t. $\preceq_{inf}$.

The action of repairing a completeness error e via the compound, repair action **RepairByDelete** allows us to simultaneously repair all the errors e' in $E_{CM}(W)$ such that $e \preceq_{inf} e'$. More formally, we can prove the following proposition.

Proposition 6.3 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $e_i \in E_{CM}(W), i = 1 \ldots n$, be completeness errors in W such that $e_1 \preceq_{inf} e_2 \preceq_{inf} \cdots \preceq_{inf} e_n$. Then, after repairing the completeness error e_1 by using the operation $W' =$* **RepairByDelete**(W, e_1)*, $e_i \notin E_{CM}(W')$, for every $i = 1 \ldots, n$.*

Proof. By Definition 4.12, a completeness error is repaired by removing (any subterm of) the XML documentthat fires the partial rewriting sequence that leads to a missing requirement.

Consider the completeness errors $e_1, \ldots, e_n \in E_{CM}(W)$ such that $e_1 \preceq_{inf} \ldots \preceq_{inf} e_n$, with $RSeq(e_i) = p_i \overset{r_i,w_i,\sigma_i}{\rightharpoonup} t_1 \rightharpoonup^* req_i$, $i = 1, \ldots, n$. By Definition 6.2, we have that $p_{i|w_i} \trianglelefteq (p_{i+1|w_{i+1}})_{|w}$, for some $w \in O_{\mathcal{T}ag}(p_{i+1|w_{i+1}})$, $i = 1, \ldots n-1$. Then, by using **RepairByDelete**(e_1, W) to repair the error e_1, the term $p_{i|w_i}$ is removed from the whole XML repository W. This fact implies that each term $(p_{i+1|w_{i+1}})_{|w}$ is removed from W too. Thus, by Definition 4.12, the errors $e_2, \ldots e_n$ are also repaired, which concludes the proof. ■

Note that Proposition 6.3 does not depend on the kind of completeness error considered (Missing XML document, Universal completeness error, and Existential completeness error).

We say that two completeness errors e_1 and e_2 are *equivalent* w.r.t. $\preceq_{inf}$ (in symbols, $e_1 \approx_{inf} e_2$) iff $e_1 \preceq_{inf} e_2$ and $e_2 \preceq_{inf} e_1$. By using the equivalence $\approx_{inf}$, we can naturally lift the preorder $\preceq_{inf}$ on $E_{CM}(W)$ to a (well-founded) partial order $\sqsubseteq_{inf}$ on $E_{CM}(W)/_{\approx_{inf}}$, that is, the set of all equivalence classes $[e]$ on $E_{CM}(W)$ w.r.t. $\approx_{inf}$.

More specifically, given $[e_1], [e_2] \in E_{CM}(W)/_{\approx inf}$, $[e_1] \sqsubseteq_{inf} [e_2]$ iff $e_1 \preceq_{inf} e_2$.

We say that an equivalence class $[e]$ is *minimal* w.r.t. $\sqsubseteq_{inf}$, if and only if there does not exist $[e'] \in E_{CM}(W)/_{\approx_{inf}}$ such that $[e'] \preceq_{inf} [e]$ and $[e'] \neq [e]$.

Now, the following results, which are both a consequence of Proposition 6.3, hold:

Proposition 6.4 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $e \in E_{CM}(W)$. By applying a* **RepairByDelete** *action to e, all the completeness errors in $[e]$ get repaired.*

Proof. Immediate. Simply observe that $e \preceq_{inf} e'$ for each e' in $[e]$. Hence —by Proposition 6.3— the repair for e also fixes the error e'. ■

Proposition 6.5 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $e_1, e_2 \in E_{CM}(W)$. If $[e_1] \sqsubseteq_{inf} [e_2]$, then, by repairing the error e_1 through a* **RepairByDelete** *action, we fix all the errors in $[e_2]$ as well.*

Proof. Since $[e_1] \sqsubseteq_{inf} [e_2]$, $e_1 \preceq_{inf} e_2$. Thus, the repair for e_1 also fixes e_2 by Proposition 6.3. This implies that all the errors in $[e_2]$ get repaired by Proposition 6.4. ■

The notion of Completion with Deletion Reduction strategy is formalized as follows.

Definition 6.6 (Completion with Deletion Reduction Strategy)
Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $[e_0], \ldots, [e_n]$ be all the minimal equivalence classes w.r.t. $\sqsubseteq_{inf}$ in $E_{CM}(W)/_{\approx_{inf}}$. Then, a Completion with Deletion Reduction *strategy for W (also called $\mathbb{MDR}(W)$) is a sequence*

$$\langle \textbf{RepairByDelete}(W_0, e_0), \ldots, \textbf{RepairByDelete}(W_n, e_n) \rangle$$

where for every $i = 0 \ldots n$, W_i is an XML repository, and $W_0 = W$.

Roughly speaking, the above strategy completes a given XML repository by fixing *only* a completeness error e for each minimal equivalence class $[e]$. Therefore, the number of **RepairByDelete** actions, that are needed to achieve completeness, is less or equal than the total number of error in $E_{CM}(W)$.

Proposition 6.7 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Then, a $\mathbb{MDR}(W)$ strategy is a completion strategy for W that transforms W into an XML repository W' such that $|\mathbb{MDR}(W)| \leq |E_{CM}(W)|$.*

Proof. Given a set of completeness error evidences $E_{CM}(W)$, let $e \in E_{CM}(W)$ be an arbitrary completeness error in $E_{CM}(W)$. Let $[e_0], \ldots, [e_n]$ be all the minimal equivalence classes w.r.t. $\sqsubseteq_{inf}$ of $E_{CM}(W)/_{\approx_{inf}}$.

It is easy to see that the partial order $\sqsubseteq_{inf}$ is well-founded, that is, for each $[e]$, there always exists a minimal $[e_i]$ such that $[e_i] \sqsubseteq_{inf} [e]$, for some $i = 0, \ldots, n$.

Now, we can distinguish two cases.

Case(i). $e \in [e_i]$ for some $i = 0, \ldots, n$. This implies that there exists a compound action **RepairByDelete**(W, e_i) in the $\mathbb{MDR}(W)$ strategy, that repairs all the errors in $[e_i]$ by Proposition 6.4. Hence, e is repaired.

Case(ii). $e \notin [e_i]$ for all $i = 0, \ldots, n$. Then, there exists a non minimal equivalence class $[\hat{e}]$ such that $e \in [\hat{e}]$. Since $\sqsubseteq_{inf}$ is well-founded, there exists a minimal class $[e_i]$, for some $i = 0, \ldots, n$, such that $[e_i] \sqsubseteq_{inf} [\hat{e}]$. As there exists the compound action **RepairByDelete**(W, e_i) in the $\mathbb{MDR}(W)$ strategy, e_i gets fixed. Therefore, by Proposition 6.5, all the errors in $[\hat{e}]$ get repaired as well. Hence, e is repaired.

Therefore, any error e is repaired by applying the $\mathbb{MDR}(W)$ strategy, that transforms W into a repository W' such that $E_{CM}(W') = \emptyset$. This implies that $\mathbb{MDR}(W)$ is a completion strategy. Also note that the number of **RepairByDelete**, appearing in the $\mathbb{MDR}(W)$ strategy, is equal to the number of minimal equivalence classes in $E_{CM}(W)/_{\approx_{inf}}$, which is lower than $|E_{CM}(W)|$. Hence, $|\mathbb{MDR}(W)| \leq |E_{CM}(W)|$ ■

Example 6.8

Consider the following XML repository W and the set of completeness rules I_{CM}:

XML repository $W = \{p_1, p_2, p_3, p_4\}$	Completeness rules $I_{CM} = \{r_1, r_2, r_3, r_4\}$
$p_1 = m(s(b), f(a))$	$r_1 = f(X) \rightharpoonup \sharp g(X)\langle E \rangle$
$p_2 = m(m(g(a)))$	$r_2 = g(X) \rightharpoonup \sharp h(X)\langle E \rangle$
$p_3 = m(l(b, a))$	$r_3 = h(X) \rightharpoonup \sharp p(X)\langle A \rangle$
$p_4 = h(b)$	$r_4 = l(X, Y) \rightharpoonup \sharp p(X, Y)\langle A \rangle$
$p_5 = m(b, h(c, b))$	

Then, $E_{CM} = \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ is the set of detected completeness errors of W that is delivered by our verification methodology, where

$e_0 = (m(m(g(a))) \rightharpoonup h(a), \{p_4\}, E)$	$e_3 = (m(s(b), f(a)) \rightharpoonup g(a) \rightharpoonup h(a), \{p_4\}, E)$
$e_1 = (h(b) \rightharpoonup p(b), W)$	$e_4 = (m(m(g(a))) \rightharpoonup h(a) \rightharpoonup p(a), W)$
$e_2 = (m(l(b, a)) \rightharpoonup p(b, a), W)$	$e_5 = (m(s(b), f(a)) \rightharpoonup g(a) \rightharpoonup h(a) \rightharpoonup p(a), W)$
$e_6 = (m(b, h(c, b)) \rightharpoonup p(b), W)$	$e_7 = (m(b, h(c, b)) \rightharpoonup p(c), W)$

Note that the errors e_1, e_2, e_4, e_5, e_6, and e_7 refer to missing XML documents, whereas e_0 and e_3 are existential completeness errors.

The preorder $\preceq_{inf}$ is defined as follows:

$$\{e_0 \preceq_{inf} e_4, e_4 \preceq_{inf} e_0, e_3 \preceq_{inf} e_5, e_5 \preceq_{inf} e_3, e_1 \preceq_{inf} e_6, e_1 \preceq_{inf} e_7\} \cup \{e_i \preceq_{inf} e_i | i = 0, \ldots, 7\}$$

where $[e_0] = \{e_0, e_4\}$, $[e_1] = \{e_1\}$, $[e_2] = \{e_2\}$, $[e_3] = \{e_3, e_5\}$, $[e_6] = \{e_6, e_7\}$. Therefore, the minimal, equivalence classes are

$$[e_0], [e_1], [e_2], [e_3].$$

Note that $[e_6]$ is not minimal as $[e_1] \sqsubseteq_{inf} [e_6]$.

Then, the $\mathbb{MDR}(W)$ strategy is

$$\langle \mathbf{RepairByDelete}(W, e_0), \mathbf{RepairByDelete}(W_1, e_1),\\ \mathbf{RepairByDelete}(W_2, e_2), \mathbf{RepairByDelete}(W_3, e_3)\rangle$$

Since the $\mathbb{MDR}(W)$ strategy is a completion strategy, by Definition 6.1 we have

$$\begin{array}{l} W_1 = \mathbf{RepairByDelete}(W, e_0) \\ W_2 = \mathbf{RepairByDelete}(W_1, e_1) \\ W_3 = \mathbf{RepairByDelete}(W_2, e_2) \\ W_4 = \mathbf{RepairByDelete}(W_3, e_3) \end{array}$$

Finally, a complete XML repository $W_4 = \{p'_1, p'_2, p'_3, p'_5\}$ is obtained, (i.e., $E_{CN}(W_4) = \emptyset$), where $p'_1 = m(s(b))$, $p'_2 = m(m(\ ))$, $p'_3 = m(\ )$, and $p'_5 = m(b)$. Note that the $\mathbb{MDR}(W)$ strategy reduces the number of errors that are needed to repair the XML repository, since it only addresses 4 errors from the original set of errors $E_{CM}(W)$.

6.2. Completion with Addition Reduction Strategy

The Completion with Addition Reduction strategy completes an XML repository W by adding all the required missing information contained in the set $E_{CM}(W)$ by means of multiple applications of a variant of the **RepairByInsert** repair action of Section 4.2.1.

The strategy uses the following preorder $\preceq^{sup}$ on $E_{CM}(W)$.

Definition 6.9 ($\preceq^{sup}$) *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $e_1, e_2 \in E_{CM}(W)$ be two completeness errors with $RSeq(e_1) = p_1 \overset{r_1,w_1,\sigma_1}{\rightharpoonup} t_1 \rightharpoonup^* req_1$ and $RSeq(e_2) = p_2 \overset{r_2,w_2,\sigma_2}{\rightharpoonup} t_2 \rightharpoonup^* req_2$. Then,*

$$e_1 \preceq^{sup} e_2 \quad \textit{iff} \quad \textit{there exists } w \in O_{\mathcal{T}ag}(req_2) \textit{ s.t. } req_1 \trianglelefteq req_{2|w}.$$

Algorithm 1 Procedure to repair a set of completeness errors ordered by $\preceq^{sup}$.

```
Require:
    $E = \{e_1, \ldots, e_m\}$ s.t. $e_1 \preceq^{sup} \ldots \preceq^{sup} e_m$
    $W$ be an XML repository
Ensure:
    $W \mid \forall e \in E, e \notin E_{CM}(W)$
procedure RepairByInsert* $(W, E)$
    $P_R = \{\}$  // Set of repaired documents.
    for $i \leftarrow m$ downto $1$ do
        if $e_i = (p \rightharpoonup^{+} req, W)$ and $P_R = \{\}$ then
            $W \leftarrow$ add$(req, W)$
            $P_R \leftarrow P_R \cup \{req\}$
        else if $e_i = (p \rightharpoonup^{+} req, P, E)$ and $P_R = \{\}$ then
            $p \leftarrow ask\_user()$  // Ask user to select an XML document of $P$
            $w \leftarrow ask\_user()$  // Ask user to select a position $w$ in $p$
            $p' \leftarrow$ insert$(p, w, req)$
            $W \leftarrow W \setminus \{p\} \cup \{p'\}$
            $P_R \leftarrow P_R \cup \{p\}$
        else if $e_i = (p \rightharpoonup^{+} req, P, A)$ then
            $P_{Aux} \leftarrow P \setminus P_R$
            for all $p \in P_{Aux}$ do
                $w \leftarrow ask\_user()$  // Ask user to select a position $w$ in $p$
                $p' \leftarrow$ insert$(p, w, req)$
                $W \leftarrow W \setminus \{p\} \cup \{p'\}$
                $P_R \leftarrow P_R \cup \{p\}$
            end for
        end if
    end for
end procedure
```

Intuitively, given two completeness errors e_1 and e_2, $e_1 \preceq^{sup} e_2$, whenever the missing requirement of e_1 is embedded into (a subterm of) the missing requirement of e_2.

The following trivial facts hold.

- **Fact 1.** Let us consider two completeness errors e_1 and e_2 such that $e_1 \preceq^{sup} e_2$. The missing requirement $Req(e_1)$ (resp., $Req(e_2)$) is the information to be added to repair the error e_1 (resp. e_2). Since $Req(e_1)$ is embedded into $Req(e_2)$, if we add $Req(e_2)$, we also add $Req(e_1)$.

- **Fact 2.** If the missing document/existential completeness error e_2 is repaired by inserting the corresponding missing requirement, then we also get all the completeness errors e_1 repaired such that $e_1 \preceq_2^{sup}$ and e_1 is a *missing XML document error* or an *existential error*. This is because, in a simple step, we add the information that embeds the missing requirements of all the considered missing XML documents and existential completeness errors.

- **Fact 3.** Let e_2 be a universal completeness error. If e_2 is repaired by adding the corresponding missing requirement, then every $e_1 \preceq_2^{sup}$ is also fixed simultaneously.

These facts are the basis of Algorithm 1, which implements **RepairByInsert***: a variant of the compound, repair action **RepairByInsert** of Definition 4.9 that allows us not only to fix a single completeness error but also multiple errors by applying a reduced number of primitive **insert/add** actions. Note that, to ensure the soundness of our technique, we require **add** and **insert** actions to be acceptable w.r.t. the XML specification and XML repository under consideration (as in the case of the original **RepairByInsert**).

Algorithm 1 takes in input an XML repository to be repaired and a set of completeness errors $\{e_1, \ldots, e_m\}$ such that $e_1 \preceq^{sup} \ldots \preceq^{sup} e_m$, and delivers an XML repository where every e_i has been fixed. The algorithm fixes the errors $e_1, \ldots, e_m$ in reverse order (that is, from e_m to e_1): initially, it repairs the error e_m by adding the required missing information $Req(e_m)$. Note that, if $e_1 \preceq^{sup} \ldots \preceq^{sup} e_m$ does not contain universal completeness errors (Fact 2) or e_m is a universal completeness error (Fact 3), it suffices to repair e_m to get every e_i fixed.

The Completion with Addition Reduction strategy is based on the partial order $\sqsubseteq^{sup}$, induced by the preorder $\preceq^{sup}$, which is constructed as follows.

Given two completeness errors e_1, e_2, e_1 and e_2 are *equivalent* w.r.t. $\preceq^{sup}$ (in symbols, $e_1 \approx^{sup} e_2$) iff $e_1 \preceq^{sup} e_2$ and $e_2 \preceq^{sup} e_1$. By using the equivalence $\approx_{sup}$, we can naturally lift the preorder $\preceq^{sup}$ on $E_{CM}(W)$ to a (well-founded) partial order $\sqsubseteq^{sup}$ on $E_{CM}(W)/_{\approx^{sup}}$, that is, the set of all equivalence classes $[e]$ on $E_{CM}(W)$ w.r.t. $\approx^{sup}$.

More specifically, given $[e_1], [e_2] \in E_{CM}(W)/_{\approx^{sup}}$, $[e_1] \sqsubseteq^{sup} [e_2]$ iff $e_1 \preceq^{sup} e_2$.

We say that an equivalence class $[e]$ is *maximal* w.r.t. $\sqsubseteq^{sup}$, if and only if there does not exist $[e'] \in E_{CM}(W)/_{\approx^{sup}}$ such that $[e] \sqsubseteq^{sup} [e']$ and $[e'] \neq [e]$.

An *error chain* is a sequence $[e_1] \sqsubseteq^{sup} [e_2] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_n]$. An *error chain* $[e_1] \sqsubseteq^{sup} [e_2] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_n]$ is *maximal* w.r.t. $\sqsubseteq^{sup}$ iff $[e_n]$ is maximal w.r.t. $\sqsubseteq^{sup}$. By notation $Set(e_1 \sqsubseteq^{sup} e_2 \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} e_n)$, we denote the set $\{e_1, \ldots, e_n\}$.

Given an error chain $[e_1] \sqsubseteq^{sup} [e_2] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_n]$, the next proposition states that to fix all the errors in each $[e_i]$, $i = 1, \ldots, n$, it suffices to consider one error e_i for each class $[e_i]$ and apply a single **RepairByInsert*** on such errors.

Proposition 6.10 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $e_1, \ldots, e_n \in E_{CM}(W)$ such that $[e_1] \sqsubseteq_{sup} \ldots \sqsubseteq_{sup} [e_n]$. Then,*

$$\textbf{RepairByInsert}^*(W, Set([e_1] \sqsubseteq^{sup} [e_2] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_n]))$$

delivers an XML repository W' in which every error $e' \in [e_i]$ is repaired, for all $i = 1, \ldots, n$.

Proof. This is an immediate consequence of Fact 2 and Fact 3. ■

Now, by exploiting 6.10, a naïve completion strategy would repair an incomplete XML repository, by executing Algorithm 1 on each sequence of errors $e_1 \preceq^{sup} e_2 \preceq^{sup} \ldots \preceq^{sup} e_n$ such that the error chain $[e_1] \sqsubseteq^{sup} [e_2] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_n]$ is maximal. However, this approach is not efficient at all. Indeed, it may require to correct the very same error e several times, as $[e]$ may occur in more than one maximal error chain. In other words, if $[e]$ occurs in two different error chains and one occurrence of e is repaired, this naïve strategy may lead to inefficacy, since the second insertion action might not be needed anymore. The idea of the Completion with Addition Reduction strategy is to distill, from the set of all maximal error chains on $E_{CM}(W)/_{\approx^{sup}}$, a set $\Gamma_{E_{CM}(W)} \subseteq 2^{E_{CM}(W)}$ such that

(i) each $K \in \Gamma_{E_{CM}}$ is a set $\{e_1, \ldots, e_k\}$ of completeness errors with $[e_1] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_k]$;

(ii) the sets of errors that belong to $\Gamma_{E_{CM}(W)}$ are pairwise disjoint (i.e., any given completeness error e belongs to *only* one set $K \in \Gamma_{E_{CM}(W)}$).

Then, Algorithm 1 is applied to every set of errors in $\Gamma_{E_{CM}(W)}$. This approach solves the issues introduced by the naïve approach, since it removes completeness error duplicates and guarantees that each considered completeness error is repaired only once.

More formally, let $\mathsf{C}^{\preceq^{sup}}_{E_{CM}(W)} = \{C_0, \ldots, C_n\}$ be the set of all maximal error chains on $E_{CM}(W)/_{\approx^{sup}}$ such that $|C_i| \geq |C_{i+1}|$ for $0 \leq i < n$. By $\mathsf{C}^{\preceq^{sup}}_{E_{CM}}(i)$, we denote the set $\{Set(C_0), \ldots, Set(C_i)\}$. Then, we define the set $\Gamma_{E_{CM}(W)}$ as follows:

$$\begin{aligned}\Gamma_{E_{CM}(W)} = \{K_i \mid K_i = \mathit{dif}(\mathsf{C}^{\preceq^{sup}}_{E_{CM}}(i)),\ \mathit{with}\ 0 \leq i \leq |\mathsf{C}^{\preceq^{sup}}_{E_{CM}}|\},\\ \text{where } \mathit{dif}(\{x_0\}) = x_0 \qquad\qquad\qquad\\ \mathit{dif}(\{x_0, \ldots, x_n\}) = (((x_n \setminus x_{n-1}) \setminus \ldots) \setminus x_0), \text{ if } n > 0\end{aligned}$$

The Completion with Addition Reduction strategy is thus formalized by using the set $\Gamma_{E_{CM}(W)}$ as follows.

Definition 6.11 (Completion with Addition Reduction Strategy, $\mathbb{MAR}$) *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Let $\Gamma_{E_{CM}(W)} = \{K_0, \ldots, K_n\}$.*

Then, a Completion with Addition Reduction *strategy for W (also called* $\mathbb{MAR}(W)$ *strategy) is a sequence*

$$\langle \mathbf{RepairByInsert}^*(W_0, K_0), \ldots, \mathbf{RepairByInsert}^*(W_n, K_n)\rangle$$

where for every $i = 0 \ldots n$, W_i is an XML repository, and $W_0 = W$.

The following proposition states that a $\mathbb{MAR}(W)$ strategy is a completion strategy that allows an XML repository W to be completed by applying a number of **RepairByInsert*** actions that is less or equal than the total number of completeness errors in W.

Proposition 6.12 *Let (R, I_{CN}, I_{CM}) be an XML specification. Let W be an XML repository, and let $E_{CM}(W)$ be the set of completeness errors of W w.r.t. I_{CM}. Then, a $\mathbb{MAR}(W)$ strategy is a completion strategy for W that transforms W into an XML repository W' such that $|\mathbb{MAR}(W)| \leq |E_{CM}(W)|$*

Proof. Given a set of completeness errors $E_{CM}(W)$, let $e \in E_{CM}(W)$ be an arbitrary completeness error in $E_{CM}(W)$. First, observe that each error chain w.r.t. $\sqsubseteq^{sup}$ is finite by Definition of $\sqsubseteq^{sup}$ and the fact that each XML document is represented by a *finite* term in $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$. Therefore, there must exist a maximal error chain $[e_1] \sqsubseteq^{sup} [e_2] \sqsubseteq^{sup} \ldots \sqsubseteq^{sup} [e_n]$ such that $e \in [e_i]$ for some $i = 1, \ldots, n$. Now, by definition of $\Gamma_{E_{CM}(W)}$, there must exists $K \in \Gamma_{E_{CM}(W)}$ such that $e_i \in K$. Since **RepairByInsert**$^*(W_i, K)$ occurs in the $\mathbb{MAR}(W)$ strategy by Definition 6.11, e_i gets repaired. Then, by Proposition 6.10, all errors in $[e_i]$ get repaired. Hence, e is repaired as well.

Therefore, any error e is repaired by applying the $\mathbb{MAR}(W)$ strategy, that transforms W in a repository W' such that $E_{CM}(W') = \emptyset$. This implies that $\mathbb{MAR}(W)$ is a completion strategy. Also note that the number of **RepairByDelete**, appearing in the $\mathbb{MDR}(W)$ strategy, is equal to the number of elements of $\Gamma_{E_{CM}(W)}$, which is less or equal than $|E_{CM}(W)|$. Hence, $|\mathbb{MAR}(W)| \leq |E_{CM}(W)|$. ■

Example 6.13 ____

Consider the XML repository W, the set of completeness rules I_{CM}, and the set of detected completeness errors $E_{CM} = \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ of Example 6.8.

The preorder $\preceq^{sup}$ is defined as follows:

$$\{e_0 \preceq^{sup} e_3, e_3 \preceq^{sup} e_0, e_1 \preceq^{sup} e_6, e_6 \preceq^{sup} e_1, e_1 \preceq^{sup} e_2, e_6 \preceq^{sup} e_2, e_4 \preceq^{sup} e_5,$$
$$e_5 \preceq^{sup} e_4, e_4 \preceq^{sup} e_2, e_5 \preceq^{sup} e_2\} \cup \{e_i \preceq^{sup} e_i | i = 0, \ldots, 7\}$$

and the equivalence classes w.r.t. $\approx^{sup}$ are

$$[e_0] = \{e_0, e_3\}, [e_1] = \{e_1, e_6\}, [e_2] = \{e_2\}, [e_4] = \{e_4, e_5\}, [e_7] = \{e_7\}.$$

The set of all maximal error chains, in reverse order w.r.t. their length, is

$$C^{\preceq^{sup}}_{E_{CM}} = \{[e_1] \sqsubseteq^{sup} [e_2], [e_4] \sqsubseteq^{sup} [e_2], [e_0], [e_7]\}.$$

The partition $\Gamma_{E_{CM}}$ is then $\Gamma_{E_{CM}} = \{\{e_0\}, \{e_4, e_2\}, \{e_1\}, \{e_7\}\}$.

Then, the $\mathbb{MAR}(W)$ strategy is $\langle$**RepairByInsert**$^*(W, \{e_0\})$, **RepairByInsert**$^*(W_1, \{e_4, e_2\})$, **RepairByInsert**$^*(W_2, \{e_1\})$, **RepairByInsert**$^*(W_3, \{e_7\})\rangle$

and it yields a complete XML repository

$$W' = \{m(s(b), f(a)), p(b, a), p(b), m(m(g(a))), m(l(b, a)), h(a, b), m(b, h(c, b)), p(c)\}$$

.

7. Implementation

The rule-based repairing methodology presented in this work has been implemented in the WifiX tool. The underlying rewriting machinery of WifiX is written in Maude and consists of about 250 Maude function definitions (approximately 5K lines of source code). WifiX also comes with an intuitive Web user interface, which allows the verification and repairing facilities to be used through a Web service. The provided verification engine uses the Web service functionality described in [15], while the repair engine has been newly reimplemented and can also work in stand-alone mode.

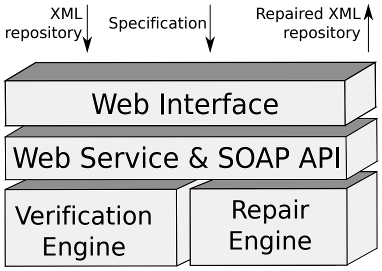


Figure 4: WifiX architecture.

The architecture of WifiX is depicted in Figure 4 and is structured into two standard layers, front-end and back-end. The back-end layer provides the Web services that are offered through the front-end layer. This architecture allows clients on the network to easily invoke the Web service functionality through the available interfaces.

WifiX consists of four main components: **Web interface**, **Web service & SOAP API**, **Verification engine** and **Repair engine**.

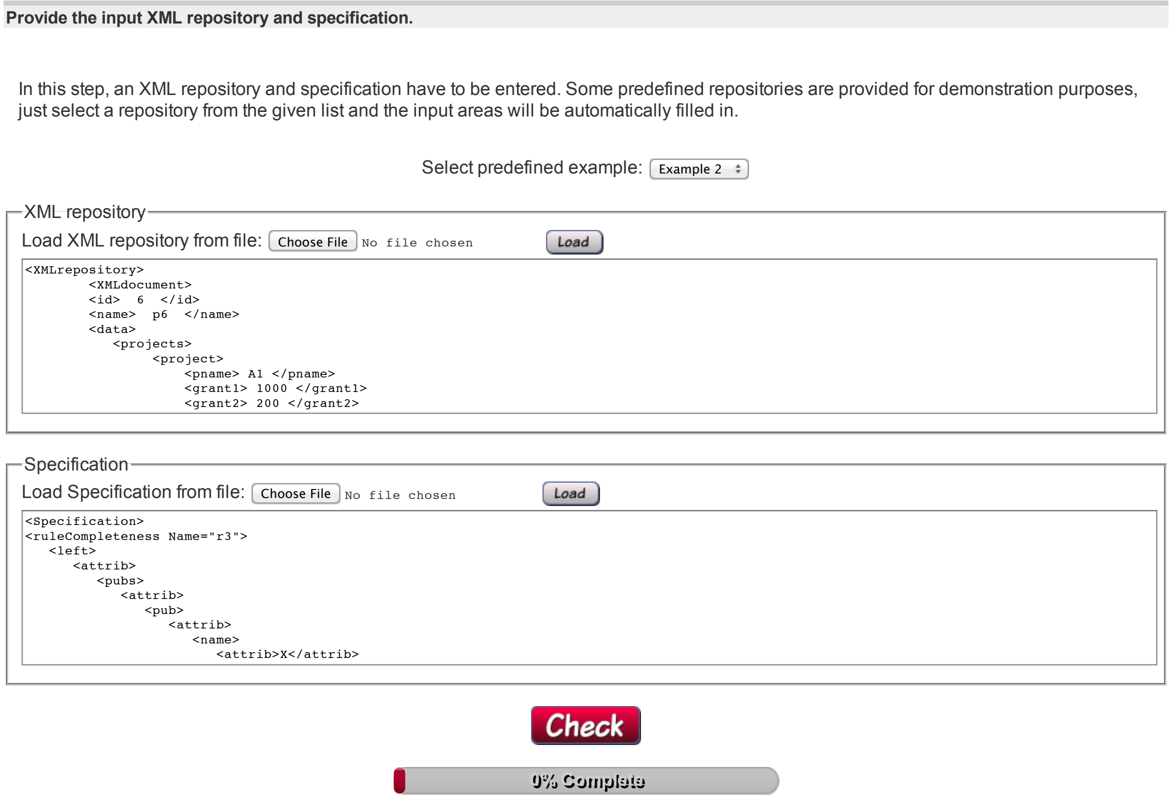
Web interface. This module is purely implemented in HTML5 and JSP. It represents the front-end layer and provides an intuitive, versatile Web user interface, which interacts with the Web service to invoke the capabilities of the verification and repair engine, and it offers the possibility to choose the repair strategy to be applied.

Web service & SOAP API. The Web service is developed by defining a SOAP API that encompasses the executable library of the core engine. This is achieved by making use of the `Apache Axis` module which is integrated into `Apache Tomcat` Web server. `Apache Axis` is an implementation of the SOAP protocol that handles all of the procedures needed for the Web service deployment. Detected errors as well as repaired XML repositories are encoded in XML in order to be transferred from the Web server to client applications by means of the SOAP protocol.

Verification engine. The verification engine module uses the Web service functionality described in [15], which is implemented in the Maude language and is independent of the other system components. It gets an XML repository and an XML specification and delivers the computed list of correctness/completeness errors as outcome.

Repair engine. The repair engine is also implemented in Maude and uses the list of errors delivered by the verification engine to fix the XML repository following the chosen repair strategy.

In Figures 5–8 we illustrate part of the repair functionality that is offered by the WifiX system. Specifically, Figure 5 shows the loading phase of the XML repository to be fixed and the considered XML specification. Figures 6–7 illustrate the selection of the correction strategy to apply, and the specification of the associated repair actions. Finally, Figure 8 partially displays the resulting XML repository.



Provide the input XML repository and specification.

In this step, an XML repository and specification have to be entered. Some predefined repositories are provided for demonstration purposes, just select a repository from the given list and the input areas will be automatically filled in.

Select predefined example: Example 2

XML repository

Load XML repository from file: No file chosen

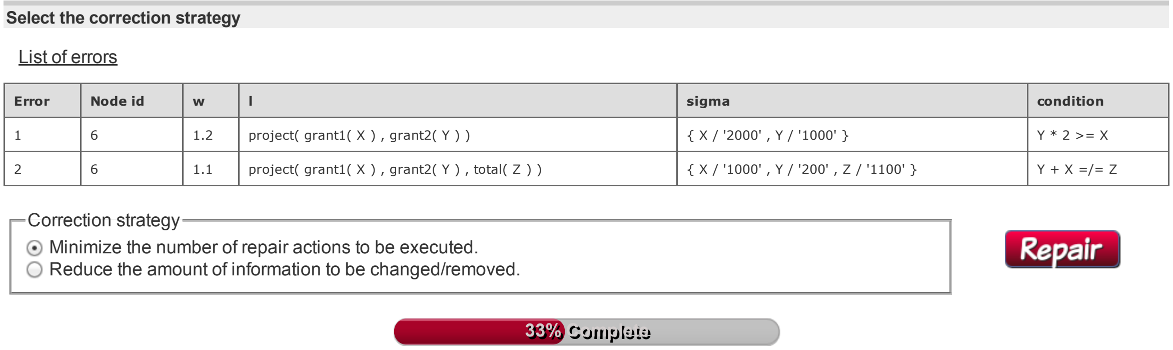
```
<XMLrepository>
        <XMLdocument>
        <id>  6  </id>
        <name>  p6  </name>
        <data>
           <projects>
                <project>
                    <pname> A1 </pname>
                    <grant1> 1000 </grant1>
                    <grant2> 200 </grant2>
```

Specification

Load Specification from file: No file chosen

```
<Specification>
<ruleCompleteness Name="r3">
   <left>
      <attrib>
         <pubs>
            <attrib>
               <pub>
                  <attrib>
                     <name>
                        <attrib>X</attrib>
```

Figure 5: Loading XML repository



Select the correction strategy

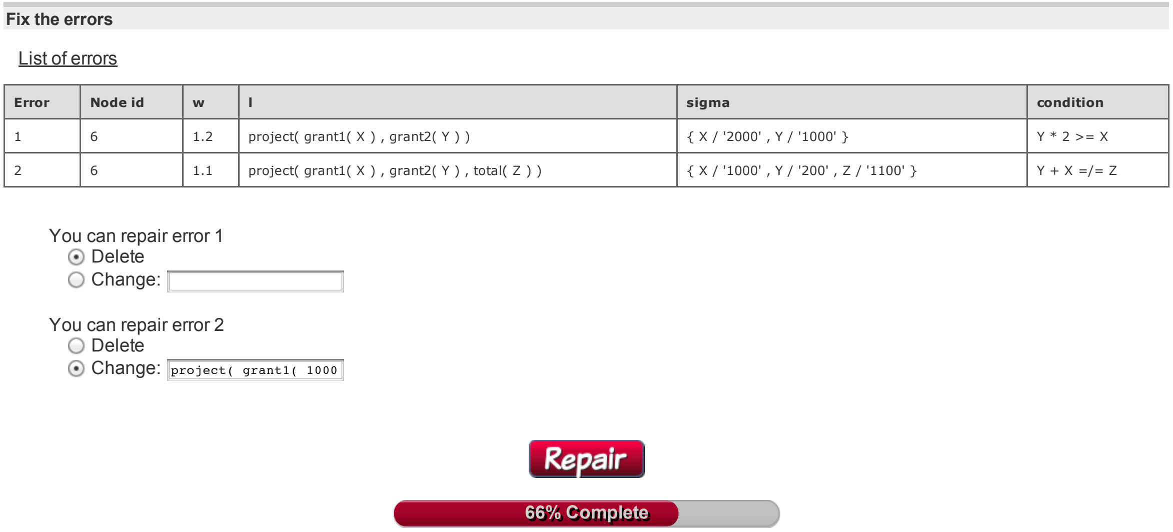
List of errors

Error	Node id	w	l	sigma	condition
1	6	1.2	project(grant1(X) , grant2(Y))	{ X / '2000' , Y / '1000' }	Y * 2 >= X
2	6	1.1	project(grant1(X) , grant2(Y) , total(Z))	{ X / '1000' , Y / '200' , Z / '1100' }	Y + X =/= Z

Correction strategy

● Minimize the number of repair actions to be executed.
○ Reduce the amount of information to be changed/removed.

Figure 6: Selecting the repair strategy



Fix the errors

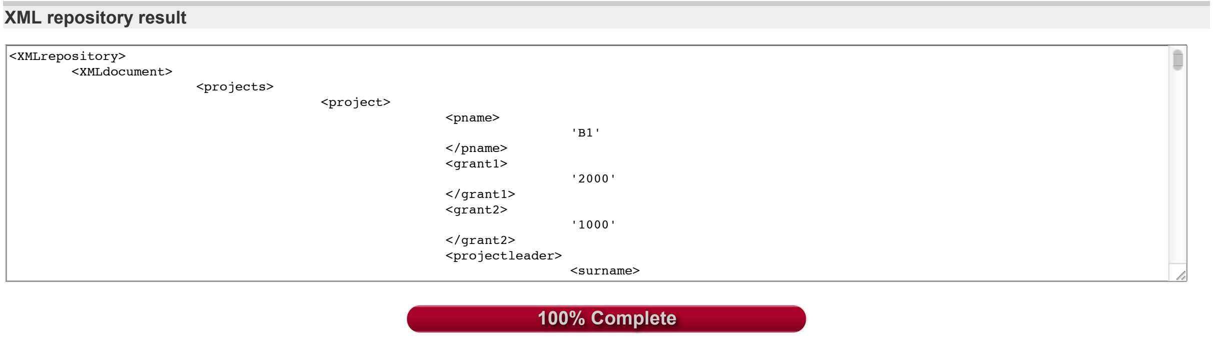
List of errors

Error	Node id	w	l	sigma	condition
1	6	1.2	project(grant1(X) , grant2(Y))	{ X / '2000' , Y / '1000' }	Y * 2 >= X
2	6	1.1	project(grant1(X) , grant2(Y) , total(Z))	{ X / '1000' , Y / '200' , Z / '1100' }	Y + X =/= Z

You can repair error 1
● Delete
○ Change:

You can repair error 2
○ Delete
● Change: project(grant1(1000

Figure 7: Selecting the error and associated repair action



XML repository result

```
<XMLrepository>
        <XMLdocument>
                <projects>
                        <project>
                                <pname>
                                        'B1'
                                </pname>
                                <grant1>
                                        '2000'
                                </grant1>
                                <grant2>
                                        '1000'
                                </grant2>
                                <projectleader>
                                        <surname>
```

Figure 8: Displaying the transformed XML repository

7.1. Experimental results

In order to evaluate the usefulness of our approach in a realistic scenario (that is, for huge XML repositories), we have benchmarked our system by repairing correctness errors as well as completeness errors of different complexity over a significant number of large XML repositories that have been generated by using the XML documents generator xmlgen available within the XMark benchmark suite [32].

Our methodology successfully tackles two main problems that commonly arise when large datasets are considered. The first problem concerns how to find the critical errors which are necessary to correct, since many of the errors might be induced by the most critical ones, similarly to the error induction phenomenon that occurs when one compiles a program written in a high-level language. Second, the number of errors in a large dataset can be very high, and the time to correct them all, or even a subset, can be quite long. So the benefits coming from our methodology are twofold. By selecting the minimal errors we have a mechanism to focus only on the critical ones, and furthermore, we reduce significantly the time to make correction/completion on the datasets. These benefits are illustrated by Table 1 reporting on our experiments.

Specifically, Table 1 shows some of the results that we obtained for the repair of four XML repositories of increasing size (ranging from 1Mb to 10Mb) w.r.t. two different XML specifications *WS1* and *WS2*. The XML specification *WS1* aims at checking the correctness of the considered repositories w.r.t. some properties formalized by means of conditional correctness rules with regular expressions and several user-defined functions. The XML specification *WS2* defines a number of critical completeness rules that identify a significant amount of missing information.

Correctness errors, detected by applying the XML specification *WS1*, have been repaired by using the $\mathbb{NAR}$ and $\mathbb{NDR}$ correction strategies; while completeness errors, produced by the execution of *WS2*, have been repaired by means of $\mathbb{MAR}$ and $\mathbb{MDR}$ completion strategies.

For each experiment, columns **Errors** and **Fixed Errors** of Table 1 respectively provide the total number of correctness/completeness errors detected in the XML repositories and the number of correctness/completeness errors fixed by each considered repair strategy ($\mathbb{NAR}$, $\mathbb{NDR}$, $\mathbb{MAR}$, and $\mathbb{MDR}$) to achieve correctness/completeness of the XML repositories. The **Rate** column shows the percentage of errors (w.r.t. the total number of errors) that have been repaired. The repair times for the considered experiments are shown in Column **Time**, and were obtained on a 2.26GHz Intel Core 2 Duo with 4Gb of RAM memory.

The system WifiX worked well in all the experiments under examination. In all cases, the XML repositories have been corrected and completed by repairing a rather small number of errors. Indeed, in the worst case, we needed to fix 11 completeness errors out of 26, which is approximately the 42% of the total number of completeness errors detected. In the best case, we only needed to fix 17 completeness errors out of 98, which is approximately the 17% of the total number of completeness errors detected. As for correctness, we achieved similar reduction rates (ranging from 19% to 38% approximately)

XML Size	XML Spec.	Errors	Fixed Errors	Strategy	Rate	Time
1 **MB**	*WS1*	21	8	$\mathbb{NAR}$	38.10 %	0.24 s
			7	$\mathbb{NDR}$	33.33 %	0.18 s
	WS2	26	11	$\mathbb{MAR}$	42.31 %	0.46 s
			9	$\mathbb{MDR}$	34.62 %	0.30 s
5 **MB**	*WS1*	49	13	$\mathbb{NAR}$	26.53 %	1.40 s
			14	$\mathbb{NDR}$	28.57 %	1.54 s
	WS2	51	14	$\mathbb{MAR}$	27.45 %	1.32 s
			15	$\mathbb{MDR}$	29.41 %	1.45 s
8 **MB**	*WS1*	71	17	$\mathbb{NAR}$	23.94 %	1.63 s
			16	$\mathbb{NDR}$	22.54 %	1.59 s
	WS2	74	16	$\mathbb{MAR}$	21.62 %	1.82 s
			18	$\mathbb{MDR}$	24.32 %	1.98 s
10 **MB**	*WS1*	96	19	$\mathbb{NAR}$	19.79 %	2.58 s
			18	$\mathbb{NDR}$	18.75 %	2.50 s
	WS2	98	17	$\mathbb{MAR}$	17.35 %	2.84 s
			19	$\mathbb{MDR}$	19.39 %	2.92 s

Table 1: WifiX Benchmarks

in the number of correctness errors to repair.

As for the repair time, our benchmarks demonstrate that repair strategies are time efficient. The elapsed times are small even for very involved errors and complex documents. For example, running the repair facility for an XML repository about 1Mb and considering an XML specification with about 20 rules took less than 1 second (480.000 rewrites per second on standard hardware).

Finally, we want to point out that the current Maude implementation of the repairing system supersedes and greatly improves our preliminary system [23], which was only able to manage correctness for small XML repositories (of about 1Mb) within a reasonable time.

8. Conclusions

The automated management of data-intensive XML repositories is an area in which rule-based technology can make a significant contribution. Today, it is widely accepted that declarative representations are a suitable way to specify the structural aspects of XML repositories as well as many forms of XML document content. Our rule specification language is simpler than the formalizations of XML schemata based on tree automata that are often used in the literature (e.g., the regular expression types [33]). As an additional advantage, the high-performance rewriting logic language Maude [34] offers an extremely powerful, automated "reasoning engine" that is unmatched by existing repair methods for XML documents.

The repairing methodology developed in this paper deals with semantic flaws that are not addressed by classical tools. The framework comes with a language for defining correctness and completeness conditions on XML repositories. Thus, our rewriting-based (verification and) repairing technique is able to get rid of forbidden/incorrect patterns and amend incomplete/missing documents in a (semi-)automatic way.

In our implementation of WifiX, we exploit the capabilities of Maude that are particularly suitable for our task, such as the built-in associative-commutative pattern matching and intensive meta-programming capabilities. We have benchmarked WifiX and obtained a very good performance in several cases. We have also proposed a service-oriented architecture that makes the repairing capabilities of the system easily accessible to Internet requests.

As future work, we plan to study new repair strategies that minimize the information to be removed.

Acknowledgement

We would like to thank the anonymous reviewers for their helpful comments.

References

[1] C. Nentwich, W. Emmerich, A. Finkelstein, Consistency Management with Repair Actions, in: Proc. of the 25th Int'l Conference on Software Engineering, ICSE'03, IEEE Computer Society, 2003, pp. 455–464.

[2] L. Capra, W. Emmerich, A. Finkelstein, C. Nentwich, XLINKIT: a Consistency Checking and Smart Link Generation Service, ACM Transactions on Internet Technology 2(2) (2002) 151–185.

[3] MessageAUTOMATION Validator, 2010. Available at: `http://www.messageautomation.com/products/validator.html`.

[4] J. Scheffczyk, P. Rödig, U. M. Borghoff, L. Schmitz, Managing Inconsistent Repositories via Prioritized Repairs, in: Proc. of the 2004 ACM Symposium on Document Engineering (DocEng '04), ACM Press, 2004, pp. 137–146.

[5] J. Scheffczyk, P. Rödig, U. M. Borghoff, L. Schmitz, S-DAGs: Towards efficient document repair generation, in: Proc. of the 2nd Int'l Conference on Computing, Communications and Control Technologies, volume 2, pp. 308–313.

[6] J. Scheffczyk, U. M. Borghoff, P. Rödig, L. Schmitz, Consistent Document Engineering: Formalizing Type-Safe Consistency Rules for Heterogeneous Repositories, in: Proc. of the 2003 ACM Symposium on Document Engineering (DocEng '03), ACM Press, 2003, pp. 140–149.

[7] L. Bertossi, J. Pinto, Specifying Active Rules for Database Maintenance, in: G. Saake, K. Schwarz, C. Türker (Eds.), 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*, Springer, 1999, pp. 112–129.

[8] E. Mayol, 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*, Springer, 1999, pp. 62–73.

[9] L. Alexandre, J. Coelho, A High-level Approach to Web Content Verification, Advan Appl., AISCces in Computer Science, Eng. & 166 (2012) 755–764.

[10] J. Coelho, M. Florido, XCentric: logic programming for XML processing, in: I. Fundulaki, N. Polyzotis (Eds.), 9th ACM International Workshop on Web Information and Data Management (WIDM 2007), Lisbon, Portugal, November 9, 2007, ACM, 2007, pp. 1–8.

[11] J. Coelho, M. Florido, Type-Based Static and Dynamic Website Verification, in: Int'l Conference on Internet and Web Applications and Services (ICIW 2007), May 13-19, 2007, Le Morne, Mauritius, IEEE Computer Society, 2007, p. 32.

[12] P. Mancarella, G. Terreni, F. Toni, Web Sites Repairing through Abduction, Electr. Notes Theor. Comput. Sci. 235 (2009) 137–152.

[13] M. Alpuente, D. Ballis, M. Falaschi, A Rewriting-based Framework for Web Sites Verification, ENTCS 124(1) (2005). Proc. of 1st Int'l Workshop on Ruled-Based Programming (RULE'04).

[14] M. Alpuente, D. Ballis, M. Falaschi, VERDI: An Automated Tool for Web Sites Verification, in: J. J. Alferes, J. Leite (Eds.), Proc. of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04), volume 3229 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 726–729.

[15] M. Alpuente, D. Ballis, M. Falaschi, Rule-based Verification of Web Sites, Software Tools for Technology Transfer 8 (2006) 565–585.

[16] M. Alpuente, D. Ballis, M. Falaschi, 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, IEEE Computer Society Press, 2006, pp. 31–40.

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

[18] M. Alpuente, D. Ballis, M. Falaschi, P. Ojeda, D. Romero, An Abstract Generic Framework for Web Site Verification, in: Proc. of the 2008 Int'l Symposium on Applications and the Internet (SAINT 2008), IEEE Computer Society, 2008, pp. 104–110.

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

[20] J. Meseguer, Twenty Years of Rewriting Logic, Journal of Algebraic and Logic Programming (2012). To appear.

[21] N-Martí-Oliet, M. Palomino, A. Verdejo, Rewriting Logic Bibliography by Topic: 1990-2011, Journal of Logic and Algebraic Programming (2012). To appear.

[22] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, All About Maude: A High-Performance Logical Framework, volume 4350 of *Lecture Notes in Computer Science*, Springer-Verlag, 2007.

[23] D. Ballis, 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, IEEE Computer Society Press, 2007, pp. 11–19.

[24] J. Coelho, B. Dundua, M. Florido, T. Kutsia, A Rule-Based Approach to XML Processing and Web Reasoning, in: P. Hitzler, T. Lukasiewicz (Eds.), Web Reasoning and Rule Systems - Fourth International Conference (RR 2010), Bressanone/Brixen, Italy, September 22-24, 2010. Proceedings, volume 6333 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 164–172.

[25] C. Kirchner, Z. Qian, P.-K. Singh, J. Stuber, Xemantics: a Rewriting Calculus-Based Semantics of XSLT, Technical Report, LORIA, 2002. Rapport de recherche A01-R-386.

[26] I. D. Baxter, F. Ricca, P. Tonella, Web Application Transformations based on Rewrite Rules, Information and Software Technology 44 (2002).

[27] M. Alpuente, D. Ballis, D. Romero, Specification and Verification of Web Applications in Rewriting Logic, in: Formal Methods, Second World Congress FM 2009, volume 5850 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 790–805.

[28] TeReSe (Ed.), Term Rewriting Systems, Cambridge University Press, Cambridge, UK, 2003.

[29] N. Dershowitz, D. Plaisted., Rewriting, Handbook of Automated Reasoning 1 (2001) 535–610.

[30] M. R. Henzinger, T. A. Henzinger, P. W. Kopke, Computing Simulations on Finite and Infinite Graphs, in: IEEE Symp. on Found. of Computer Science, pp. 453–462.

[31] M. Leuschel, Homeomorphic Embedding for Online Termination of Symbolic Methods, in: The Essence of Computation, volume 2566 of *Lecture Notes in Computer Science*, pp. 379–403.

[32] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, R. Busse, XMark: A benchmark for xml data management, in: VLDB 2002, Proceedings of 28th Int'l Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China, pp. 974–985.

[33] H. Hosoya, B. Pierce, Regular Expressions Pattern Matching for XML, in: Proc. of 25th ACM SIGPLAN-SIGACT Int'l Symp. POPL, ACM, 2001, pp. 67–80.

[34] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, The maude 2.0 system, in: Rewriting Techniques and Applications (RTA 2003), number 2706 in Lecture Notes in Computer Science, pp. 76–87.