Verifying Integrity Constraints on Web Sites

Mary Fernández
AT&T Research
180 Park Ave.
Florham Park, NJ 07932 USA
mff@research.att.com

Daniela Florescu
INRIA
BP.105 Rocquencourt
Le Chesnay cedex, France
dana@rodin.inria.fr

Alon Levy
Dept. of Computer Science
University of Washington
Seattle, WA. 98195 USA
alon@cs.washington.edu

Dan Suciu
AT&T Research
180 Park Ave.
Florham Park, NJ 07932 USA
suciu@research.att.com

Abstract

Data-intensive Web sites have created a new form of knowledge base, as richly structured bodies of data. Several novel systems for creating data-intensive Web sites support declarative specification of a site's structure and content (i.e., the pages, the data available in each page, and the links between pages). Declarative systems provide a platform on which AI techniques can be developed that further simplify the tasks of constructing and maintaining Web sites. This paper addresses the problem of specifying and verifying integrity constraints on a Web site's structure. We describe a language that can capture many practical constraints and an accompanying sound and complete verification algorithm. The algorithm has the important property that if the constraints are violated, it proposes fixes to either the constraints or to the site definition. Finally, we establish tight bounds on the complexity of the verification problem we consider.

1 Introduction

Data-intensive Web sites have created a new form of knowledge base. They typically contain and integrate several bodies of data about the enterprise they are describing, and these bodies of data are linked into a rich structure. For example, a company's internal Web site may contain data about its employees, linked to data about the products they produce and/or to the customers they serve. The *data* in a Web site and the *structure* of the links in the site can be viewed as a richly structured knowledge base.

The management of data-intensive Web sites has received significant attention in the database community [Fernandez *et al.*, 1998; Atzeni *et al.*, 1998; Arocena and Mendelzon, 1998; Cluet *et al.*, 1998; Paolini and Fraternali, 1998]. The key insight of recent systems is to specify the structure and content of sites declaratively. These systems separate and provide direct support for the three primary steps of site creation: (1) identifying and accessing the data served at the site, (2) defining the site's structure (i.e., the pages, the data in each page, and the links between pages), and (3) specifying the HTML rendering of the site's pages. Step 2 is usually supported by a declarative, specification language.

Web-site management systems based on declarative representations offer several benefits. First, since a site's structure and content are defined declaratively, not procedurally by a program, it is easy to create multiple *versions* of a site. For example, it is possible to build internal and external views of an organization's site or to build sites tailored to novice or expert users. Currently, creating multiple versions requires writing multiple sets of programs or manually creating different sets of HTML files. Second, these systems support the evolution of a site's structure. For example, to reorganize pages based on frequent usage patterns or to extend the site's content, we simply rewrite the site's specification. Another advantage is efficient update of a site when its data sources change.

Declarative Web-site management systems also allow us to view a site's definition and its content as a knowledge base. A natural next step is to consider how reasoning techniques can further improve the process of building and maintaining Web sites. We consider the reasoning problem of verifying integrity constraints over Web sites. Specifically, when the structure of a site becomes complex, it is hard for a designer to ensure that the site will satisfy a set of desired properties. For example, we may want to enforce that all pages are reachable from the root, every organization homepage points to the homepages of its sub-organizations, or proprietary data is not displayed on the external version of the site. A study on the usability of on-line stores [Lohse and Spiller, 1998] provides other constraints that if followed, would improve the site design.

For a verification tool to be useful, it must verify constraints against a site definition, *not* a particular instance of the site, because (1) we do not want to verify the constraints every time the site instance changes, and (2) if a Web site is dynamically generated, an instance is never completely materialized making it is impossible to check the constraints. Verifying the constraints on the site definition ensures that as long as the site is generated according to the definition, the constraints will be satisfied. For this reason, the verification problem requires reasoning, and not just applying a procedure to the site. Furthermore, when the integrity constraints are not verified, the system should automatically propose a set of candidate modifications to the site definition. This raises a search problem in the space of possible modifications.

This paper makes the following contributions. First, we identify an important class of integrity constraints relevant to Web sites. Second, we describe a sound and complete algorithm for verifying the integrity constraints and an analysis of their complexity. The key feature of our algorithms is that they consider only the *specification* of the site's structure and content, not a particular instance of a site. Hence, the verification is independent of changes to the underlying site, as long as they are generated by the same specification. Finally, in cases where the verification algorithm shows that the constraints may be violated, it proposes a set of corrections to the Web site's definition.

The problem we consider is closely related to the problem of knowledge-base verification (see [VVT'98, 1998] for a recent workshop). We follow the paradigm proposed in [Levy and Rousset, 1998], where algorithms for verification are based on query containment. However, whereas in [Levy and Rousset, 1998] there was a 1-1 translation between the verification problem and query containment, a challenge in our case is to perform the appropriate transformation.

We believe that Web-site management tools based on declarative specifications will pose several important AI research problems in the near future. Hence, one of the contributions of this paper is to bring the problem to the attention of our community. In the last section, we mention other research problems in this context.

2 Declarative Management of Web Sites

Declarative systems for Web-site management are based on the principle of separating three tasks: (1) the management of the data underlying the site, (2) the definition of the site's structure and the content, and (3) the graphical presentation of the site. The first step requires identifying the sources that contain the site's data. We refer to this data as the *raw data*. These sources may include databases, structured files, or pre-existing sites. We assume that we interact with each of these sources via a *wrapper* program that produces the necessary data in tabular form. Here, we assume that the raw data is stored in a single relational database system. In the rest of the paper, we use an example that is a small fragment of a publication's Web site. Fig. 1 contains the schema of the raw data and sample data.

The second step in building a Web site requires specifying the site's structure. We describe a formalism for specifying this structure that captures features common to many declarative systems for Web-site management [Fernandez *et al.*, 1998; Atzeni *et al.*, 1998; Arocena and Mendelzon, 1998; Cluet *et al.*, 1998; Paolini and Fraternali, 1998]. We emphasize that the declarative specification is concerned with the *logical* model of the site as a set of nodes and links, not its *graphical* presentation (i.e., how each node is translated to HTML). When using the systems above, the site designer also specifies the graphical presentation of each page, usually by a set of HTML templates, each of which applies to a group of related pages.

Person

persid	*name*
p1	"john"
p2	"mary"

Article

artid	*title*	*year*
a1	"title1"	1995
a2	"title2"	1998
a3	"title3"	1998

Author

persid	*artid*
p1	a1
p1	a2
p2	a2
p2	a3

PsFile

artid	*psfile*
a2	"file2.ps"
a3	"file3.ps"

Figure 1: The schema and data underlying the publication Web site.

2.1 Specifying Web-site Structure

In order to specify a site's structure, we need to state (1) what pages exist (2) what data is available in each page, and (3) what links exist between pages. We specify the structure of site in a *site definition.* Given a site definition and a database instance, applying the definition to the database produces an instance of the site, called a *site graph.* Fig. 2 contains our example site definition and Fig. 3 contains the resulting site graph.

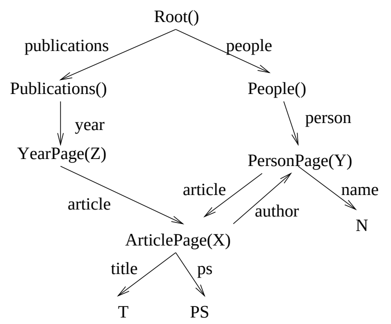

Figure 2: The site definition for our example site.

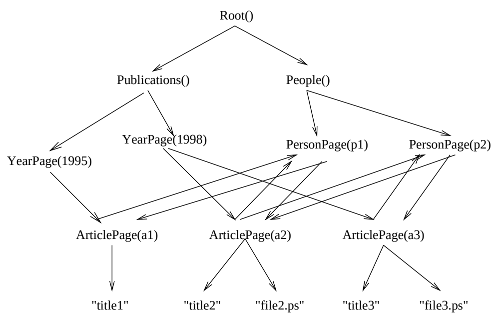

Figure 3: The example site graph.

A site definition is a graph whose nodes are labeled by variables or by functional terms of the form $f(\bar{X})$, where

$\bar{X}$ is a (possibly empty) tuple of variables. Functional nodes in the site definition represent *sets* of pages in the site graph. In our example, the node PersonPage(Y) represents the set of pages PersonPage(p) where p is a constant in the database. Non-functional nodes are leaves and have one incoming edge. They represent the data contained in the page that points to them. For example, the node N represents the name of the person Y. Functional nodes that have no arguments represent unique pages, such as the root.

Each functional node is labeled with a Horn rule that defines the conditions for the existence of instances of the node. A rule's head is an atom of the form Node(f(X)), where Node is a special predicate. For example, the rule for YearPage specifies that there will be a node for a year Z if some article was published in year Z. The rules for our example site are:

$Node(Root()) : -true.$
$Node(Publications()) : -true.$
$Node(People()) : -true.$
$Node(YearPage(Z)) : -Article(\_, \_, Z).$
$Node(PersonPage(Y)) : -Person(Y, \_).$
$Node(ArticlePage(X)) : -Article(X, \_).$

Edges in the site definition represent sets of links in the site graph. Each edge has an associated a Horn rule, which specifies the conditions for existence of a link between instances of source and destination nodes. The Horn rules use the special predicate Link. For example, the rule fourth below specifies that there is a link in the site graph between the page PersonPage(Y) and the page ArticlePage(X) if Y is an author of paper X. The third argument of the predicate Link is the link's label in the site graph.[1] We assume that in all of the rules this argument is always a constant. The rules for the links in our example are given below.

$Link(Root(), Publications(), \text{“publications”}) : -true.$
$Link(Root(), People(), \text{“people”}) : -true.$
$Link(Publications(), YearPage(Z), \text{“year”}) : -Article(\_, \_, Z).$
$Link(People(), PersonPage(Y), \text{“person”}) : -Person(Y, \_).$
$Link(PersonPage(Y), ArticlePage(X), \text{“article”}) : -$
$Author(Y, X), Article(X, \_, \_).$
$Link(YearPage(Z), ArticlePage(X), \text{“article”}) : -$
$Article(X, \_, Z).$
$Link(ArticlePage(X), PersonPage(Y), \text{“author”}) : -$
$Author(Y, X), Person(Y, \_).$

Finally, the data contained in each page is also specified by Horn rules. For every leaf associated with a functional node, we associate a Horn rule defining the contents of the leaf. The first rule below specifies that the name of a person will be contained in the appropriate person page:

$Link(PersonPage(Y), N, \text{“name”}) : -Person(Y, N).$
$Link(ArticlePage(X), T, \text{“title”}) : -Article(X, T, \_).$
$Link(ArticlePage(X), PS, \text{“ps”}) : -Article(X, \_, \_),$
$PsFile(X, PS).$

[1] This string denotes the name of the relationship between the nodes in the site graph, and not the anchor that will appear on the link in the actual site. Anchors are omitted for clarity.

Declarative specification of a Web site offers many advantages: rapid modification of the site's structure; creation of multiple versions of the site for different classes of users; and, as we explore next, the ability to reason globally about the site's structure. In principle, restructuring a site or building another version requires modifying the set of rules that define the site, instead of modifying each page and its hard-wired links.

3 Specifying Integrity Constraints

Although declarative specification can simplify the task of creating complex sites, the specification of a richly structured site can be long. For example, the specification of a customer-billing site using the Strudel specification language [Fernandez *et al.*, 1998] is 474 lines. The specification is more concise than the equivalent implementation in a scripting language, but still too large to determine without automated reasoning whether global constraints on the site are satisfied. [Lohse and Spiller, 1998] describes Web sites for on-line stores. They argue that enforcing integrity constraints on such sites is critical to customer satisfaction and describe a set of such constraints. Our goal is to take advantage of a site's declarative definition and develop algorithms for verifying that a given definition only produces sites that satisfy the given set of constraints. For our example, some possible constraints include:

IC1: All article pages are reachable from the root page.

IC2: For every article, there is a link from its article page to its PostScript source.

IC3: If two articles have a common author, there is a path between the corresponding article pages.

IC4: If two articles have been published in the same year, there is a path between the corresponding article pages.

We may also want to specify constraints that limit the length of a path between two nodes, or that force every path to a node to go through some distinguished set of nodes. We define our language for specifying these kinds of integrity constraints and formally define the verification problem.

Integrity constraints express properties we would like the Web site to have. Since the Web site is modeled as a graph, integrity constraints should be able to express the existence of certain paths between pages in the site. We express such paths using *regular-path expressions*. A regular-path expression over the set of constants $\mathcal{C}$ is formed by the following grammar (R, R_1 and R_2 denote regular-path expressions):

$$R := a \mid \mathsf{not}\ (a) \mid \_ \mid (R_1.R_2) \mid (R_1 \mid R_2) \mid R^+.$$

In the grammar, a denotes a constant in $\mathcal{C}$; not (a) matches any constant in $\mathcal{C}$ different from a. An _ denotes any constant in $\mathcal{C}$; a period denotes concatenation, and | denotes alternation. R^+, denotes 1 or more repetitions of R. For example, $a.b.\_.c^+$ denotes the set of paths beginning with ab, then an arbitrary element of $\mathcal{C}$ and then any number of occurrences of c. We use $*$ as a shorthand for $(\_)^+$, meaning an arbitrary path of length 1 or more.

Regular-path expressions are used in *path atoms* of the form $X \rightarrow R \rightarrow Y$, where R is a regular-path expression, and X and Y are terms. The atom $X \rightarrow R \rightarrow Y$ is satisfied in a labeled directed graph G by each pair of nodes X, Y for which there is path from X to Y that satisfies the regular path expression R.

In principle, we can express integrity constraints using arbitrary formulas in first-order logic. However, our main goal here is to identify a more restricted language for which it is possible to develop sound and complete verification algorithms and which is expressive enough to model integrity constraints that are of practical interest. We consider integrity constraints that have the form $\phi \Rightarrow \psi$, where ϕ and ψ are conjunctions of path atoms, atoms of the relations of the raw data, and atoms of the relation Node. Variables that appear in both ϕ and ψ are assumed to be universally quantified, while the others are existentially quantified. The following sentences express the integrity constraints in our example.

IC1: $Article(X, \_, \_) \Rightarrow Root() \rightarrow * \rightarrow ArticlePage(X)$

IC2: $Article(X, \_, \_) \Rightarrow ArticlePage(X) \rightarrow \text{“}ps\text{”} \rightarrow Y$

IC3: $Article(X, \_, \_), Article(Y, \_, \_),$
$Author(Z, X), Author(Z, Y) \Rightarrow$
$ArticlePage(X) \rightarrow * \rightarrow ArticlePage(Y)$

IC4: $Article(X, \_, Z), Article(Y, \_, Z) \Rightarrow$
$ArticlePage(X) \rightarrow * \rightarrow ArticlePage(Y)$

Given a particular site graph, it is straightforward to test whether an integrity constraint holds. However, our goal is to verify at the *intentional level* whether an integrity constraint is guaranteed to hold, i.e., given a site definition $\mathcal{R}$, test whether the integrity constraint will hold for *all* Web sites that can be generated by $\mathcal{R}$, for any possible database state. Formally, our problem is the following.

Definition 1: *Let $E_1, \ldots, E_n$ be the relations in the schema of the raw data, $\mathcal{R}$ be a site definition. Let IC be an integrity constraint. We say that $\mathcal{R}$ satisfies IC if, for any given extension $\mathcal{I}$ of the relations $E_1, \ldots, E_n$, IC is satisfied in the site graph resulting from $\mathcal{R}$ and $\mathcal{I}$.*

In our example, IC1 is satisfied, because every article has a year of publication, and therefore is reachable through the YearPage. Similarly, IC3 is also satisfied. IC2 is not satisfied, because some articles may not have PostScript sources. Although IC4 is satisfied by the site graph in Fig. 3, it is not necessarily satisfied for *every* site graph.

Next, we describe a sound and complete verification algorithm, and show how the complexity of the verification problem changes with the form of the integrity constraints considered.

4 Verification Algorithm

The crucial step of our verification algorithm is to translate the integrity constraint $\phi \Rightarrow \psi$ into a pair of Datalog programs Q_ϕ and Q_ψ. Datalog [Ullman, 1997] is a database query language where queries are specified by sets of Horn rules, and the meaning of the query is given by the least fixpoint model of the database and the rules. Our translation has the property that the integrity constraint is satisfied if and only if the datalog program Q_ϕ is *contained* in the program Q_ψ. Informally, given two queries Q_1 and Q_2 the query Q_1 *contains* the query Q_2 if Q_1's result is a superset of Q_2's result for *any* database instance. Algorithms for query containment have been studied extensively in the database literature [Ullman, 1997]. These algorithms can be viewed as logical-entailment algorithms for specific classes of logical sentences, which is why they are useful in our context.

Our algorithm has two steps.

1. Given the integrity constraint, $\phi \Rightarrow \psi$, and the site definition, $\mathcal{R}$, create a pair of Datalog queries Q_ϕ and Q_ψ.
2. We use an extended query containment algorithm to test whether Q_ϕ is contained in Q_ψ. If the containment holds, then the integrity constraint is guaranteed to hold. If not, the containment algorithm returns a set of *candidate fixes*.

We describe each step in more detail.

The algorithm in Fig. 4 translates either ϕ or ψ into a Datalog program. This step relies heavily on the possible paths specified in the structure of the site definition in order to generate Q_ϕ and Q_ψ. The subtle part of the translation concerns the path atoms. Given a path atom $X \rightarrow R \rightarrow Y$, the translation builds in a bottom-up fashion a Datalog program that defines a relation corresponding to each of the subexpressions of R. The translation varies slightly depending on whether X and Y are variables, functional terms, and whether there is another conjunct of the form $Node(X)$ ($Node(Y)$). In the figure, we show only the case when X and Y are unary functional terms.

If our extended query-containment algorithm reports that Q_ϕ is contained in Q_ψ, then then the integrity constraint is guaranteed to hold. Otherwise the containment algorithm returns a set of candidate fixes. The algorithm considers four kinds of fixes:

- Add conditions to ϕ in the integrity constraint,
- Remove conditions from the rules in the site definition $\mathcal{R}$,
- Modify $\mathcal{R}$ by adding back arcs in the site definition, and
- Suggest a set of integrity constraints to enforce on the raw data, which guarantee that the constraints on the site will hold.

The fixes are reported to the site designer, who can then decide how to proceed. Due to space limitations, we only illustrate this phase of the algorithm through the example below. Intuitively, the fixes are generated by searching through the possible modifications to Q_ϕ and Q_ψ such that for the modified queries, the containment holds.

Algorithm `IC-translate`(τ, $\mathcal{R}$, $\bar{X}$)
Input: τ is either the LHS or RHS of an IC.
$\mathcal{R}$ is the site definition.
$\bar{X}$ are the universally quantified variables in the IC.
Output: a Datalog program defining the relation $Q_\tau(\bar{X})$.

Let τ be of the form $A_1, \ldots, A_m$.
For $1 \leq i \leq m$, let P_{A_i} be the set of Horn rules returned by
  `atomToProg`(A_i, $\mathcal{R}$), with query predicate q_{A_i},
return `atomToProg`(A_1, $\mathcal{R}$) $\cup \ldots \cup$ `atomToProg`(A_m, $\mathcal{R}$)
  and the rule $Q_\tau(\bar{X}) : -q_{A_1}(\bar{X}_1), \ldots, q_{A_m}(\bar{X}_m)$
where
  if A_i is of the form $p(\bar{Y})$ **then** $\bar{X}_i = \bar{Y}$
  if A_i is of the form $f(X) \rightarrow R \rightarrow g(Y)$, **then** $\bar{X}_i = (X, Y)$.
end `IC-translate`.

// $A(\bar{X})$ is an atom; $\mathcal{R}$ is a site definition.
Algorithm `atomToProg`(A, $\mathcal{R}$)
if A is of the form $p(\bar{X})$
  then return the Datalog program: $q_A(\bar{X}) : -p(\bar{X})$.
if A is of the form $f(X) \rightarrow R \rightarrow g(Y)$ **then**
  return the Datalog program constructed as follows:
  for every rule $r \in \mathcal{R}$ of the form
  $Link(f_1(X_1), f_2(X_2), \text{“}a\text{”}) : -body$
  where the rules for f_1 and f_2 have bodies
  $body_1, body_2$ respectively, add the following rule:
    $a(f_1, X_1, f_2, X_2) : -body, body_1, body_2$
  Define an IDB predicate for R by structural induction on R:
  if R is of the form “a”, **then**
    $Q_a(f_1, X_1, f_2, X_2) : -a(f_1, X_1, f_2, X_2)$.
  if R is of the form $R_1.R_2$ **then**
    $Q_{R_1.R_2}(f_1, X_1, f_2, X_2) : -$
      $Q_{R_1}(f_1, X_1, f_3, X_3), Q_{R_1}(f_3, X_3, f_2, X_2)$
  if R is of the form $R_1 \mid R_2$ **then**
    $Q_{R_1|R_2}(f_1, X_1, f_2, X_2) : -Q_{R_1}(f_1, X_1, f_2, X_2)$.
    $Q_{R_1|R_2}(f_1, X_1, f_2, X_2) : -Q_{R_2}(f_1, X_1, f_2, X_2)$.
  if R is of the form R^* **then**
    $Q_{R^*}(f_1, X_1, f_2, X_2) : -Q_R(f_1, X_1, f_2, X_2)$.
    $Q_{R^*}(f_1, X_1, f_2, X_2) : -$
      $Q_R(f_1, X_1, f_3, X_3), Q_{R^*}(f_3, X_3, f_2, X_2)$.
the query predicate of the Datalog program is defined by:
$q_A(X, Y) : -Q_R(f, X, g, Y)$.
end `atomToProg`

Figure 4: Algorithm for translating the LHS or RHS of an integrity constraint into a Datalog program.

Consider the constraint IC1 in our example, that requires a path from the root page to any article page. The translation step produces the following two Datalog programs, whose query predicates are Q_{lhs} and Q_{rhs}. Since the RHS of the constraint involves a path atom with the regular expression $*$, the Datalog program of Q_{rhs} defines a predicate $Q_*(X_1, f_1, X_2, f_2)$ (the transitive closure of $Q\_$), which describes the possible paths in the site graph between nodes of the form $f_1(X_1)$ and $f_2(X_2)$. We also use the rules defining Q_* in the other parts of the example.

$Q_{lhs}(X) : -Article(X, \_, \_)$.

$Q_{rhs}(X) : -Q_*(\text{“}Root\text{”}, [], \text{“}ArticlePage\text{”}, X)$.
$Q_*(X, Y, Z, W) : -Q_*(X, Y, T, R), Q\_(T, R, Z, W)$.
$Q_*(X, Y, Z, W) : -Q\_(X, Y, Z, W)$.
$Q\_(\text{“}Root\text{”}, [], \text{“}Publications\text{”}, []) : -true$.
$Q\_(\text{“}Root\text{”}, [], \text{“}People\text{”}, []) : -true$
$Q\_(\text{“}Publications\text{”}, [], \text{“}YearPage\text{”}, X) : -Article(\_, \_, X)$.
$Q\_(\text{“}People\text{”}, [], \text{“}PersonPage\text{”}, X) : -Person(X, \_)$.
$Q\_(\text{“}YearPage\text{”}, X, \text{“}ArticlePage\text{”}, Y) : -Article(Y, \_, X)$.
$Q\_(\text{“}PersonPage\text{”}, X, \text{“}ArticlePage\text{”}, Y) : -Person(X, \_),$
    $Author(X, Y), Article(Y, \_, \_)$.
$Q\_(\text{“}ArticlePage\text{”}, X, \text{“}PersonPage\text{”}, Y) : -Person(Y, \_),$
    $Author(Y, X), Article(X, \_, \_)$.

Since the containment check will show that Q_{lhs} is contained in Q_{rhs}, the verification test succeeds. For IC2, the algorithm produces the following two programs, for which the containment fails.

$Q_{lhs}(X) : -Article(X, \_, \_)$.
$Q_{rhs}(X) : -Q_{ps}(\text{“}ArticlePage\text{”}, X, \_, Y)$.
$Q_{ps}(\text{“}ArticlePage\text{”}, X, \_, Y) : -Article(X, \_, \_), PsFile(X, Y)$.

However, in this case, the algorithm will propose a correction to the integrity constraint, namely adding the conjunct $PsFile(X, Y)$ to the left hand side (meaning that the constraint needs to hold only on articles that have a PostScript source).

Finally, IC4 would result in the two programs:

$Q_{lhs}(X, Y) : -Article(X, \_, Z), Article(Y, \_, Z)$.
$Q_{rhs}(X, Y) : -Q_*(\text{“}ArticlePage\text{”}, X, \text{“}ArticlePage\text{”}, Y)$.

In this case, the containment does not hold because paths between article pages in the site only go through the author pages, not through the year pages. Hence, the algorithm will suggest to add a link from ArticlePage to either Root(), the Publications(), or to the corresponding YearPage, and would propose the appropriate query to put on the new link.

5 Complexity of Verification

The algorithm described in the previous section provides a sound and complete verification algorithm in many important cases. This section characterizes these cases and establishes the complexity of the algorithm and of the verification problem. Note that in all of the results, the complexity is measured in the size of the site definition and *not* the size of the underlying raw data.

The following theorem considers the case in which there are no cycles in the nodes in the site definition.

Theorem 1: *Let $\mathcal{R}$ be a site definition and IC be an integrity constraint of the form $\phi \Rightarrow \psi$. Assume that there are no cycles between nodes in the site definition. Then, our verification algorithm is sound and complete and runs in non-deterministic polynomial time. The verification problem under these conditions is NP-complete.*

The following theorem permits cyclic site definitions, but requires that the left-hand side of the integrity constraint does not contain path atoms with Kleene star.

This is a common case, because the left-hand side usually refers to conditions on the raw data, not on the site graph.

Theorem 2: *Let $\mathcal{R}$ be a site definition and IC be an integrity constraint of the form $\phi \Rightarrow \psi$, where ϕ does not contain path atoms with Kleene star. Then, our verification algorithm is sound and complete and it runs in non-deterministic polynomial time. The verification problem under these conditions is NP-complete.*

The proof of the theorems is based on the fact that the size of Q_ϕ and Q_ψ is polynomial in the size of $\mathcal{R}$, and the complexity of the corresponding containment algorithms. Note that in general, containment of arbitrary recursive Datalog is undecidable [Shmueli, 1993], but in the cases considered above Q_ϕ is always nonrecursive. Note that if $\mathcal{R}$ contains the interpreted predicates $\leq, <, \neq$, then the complexity of the problems in the theorems is Π_2^p-complete.

6 Conclusions and Related Work

Web-site management systems based on declarative representations offer many opportunities for applying AI research to improve the Web-site construction and maintenance process. This paper considered the first such problem, namely the specification and verification of integrity constraints. We described a language for specifying a wide class of constraints and a sound and complete algorithm for verification. In addition, our algorithm suggests fixes to the site definition when the integrity constraint does not hold.

Our work can be viewed as an extension of verification methods for rule-based knowledge-base systems. Of that work, the most related to ours is [Levy and Rousset, 1998] which first showed how to use query containment techniques for knowledge-base verification. In contrast to that work, where there was a direct mapping from the knowledge base to a query containment problem, an added challenge in our context is to develop the translation to containment. [Schmolze and Snyder, 1997] considers the verification problem where rules may have side-effects, but those to not appear in our context. [Rousset, 1997] proposes an *extensional* approach to verifying constraints on snapshots of Web sites (i.e., directly on the site graphs).

Finally, we mention two additional opportunities for new AI problems in this context. The first, a generalization of the work we described here, is to specify the structure of Web sites at an even higher level. Whereas in our work we only checked whether certain integrity constraints hold for a given site definition, there may be cases that we would want to specify *only* integrity constraints for the site. The system would then consider the constraints and would propose a definition of the structure for the Web site. The challenge is to choose among multiple structures that satisfy the given constraints.

The second problem concerns automatically restructuring Web sites. The short experience in building Web sites has already shown that it is a highly iterative process. Even after the Web site is up, designers will frequently want to restructure it after understanding the patterns with which users browse the site. Perkowitz and Etzioni [Perkowitz and Etzioni, 1997] have proposed the notion of *adaptive* Web sites that restructure themselves automatically. We argue that declarative representations of Web sites provide a basis on which to build adaptive Web site techniques. In particular, once we have a model of a Web site, we can analyze the user browsing patterns and propose meaningful ways to restructure the model, and hence the site itself.

References

[Arocena and Mendelzon, 1998] Gustavo Arocena and Alberto Mendelzon. WebOQL: Restructuring documents, databases and webs. In *Intl. Conf. on Data Engineering (ICDE)*, Orlando, Florida, 1998.

[Atzeni *et al.*, 1998] P. Atzeni, G. Mecca, and P. Merialdo. Design and maintenance of data-intensive web sites. In *Conf. on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.

[Cluet *et al.*, 1998] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion. In *SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[Fernandez *et al.*, 1998] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[Levy and Rousset, 1998] A. Levy and M. Rousset. Verification of knowledge bases based on containment checking. *Artificial Intelligence*, 101(1-2):227–250, 1998.

[Lohse and Spiller, 1998] Gerald Lohse and Peter Spiller. Electronic shopping. *Comm. of the ACM*, 41(7), July 1998.

[Paolini and Fraternali, 1998] P. Paolini and P. Fraternali. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications. In *Conf. on Extending Database Technology (EDBT)*, 1998.

[Perkowitz and Etzioni, 1997] Mike Perkowitz and Oren Etzioni. Adaptive web sites: an AI challenge. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[Rousset, 1997] Marie-Christine Rousset. Verifying the web: a position statement. In *Proceedings of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV-97)*, 1997.

[Schmolze and Snyder, 1997] J. Schmolze and W. Snyder. Detecting redundant production rules. In *Proc. of the National Conference on Artificial Intelligence*, 1997.

[Shmueli, 1993] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[Ullman, 1997] Jeffrey D. Ullman. Information integration using logical views. In *Intl. Conf. on Database Theory (ICDT)*, Delphi, Greece, 1997.

[VVT'98, 1998] *Proceedings of the AAAI Workshop on Verification and Validation of Knowledge-Based Systems*, Madison, Wisconsin, July 1998.