INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians
Universität
München
LMU

An Entailment Relation for Reasoning on the Web

François Bry and Sebastian Schaffert

Technical Report, Computer Science Institute, Munich, Germany
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2003-5, May 2003

An Entailment Relation for Reasoning on the Web

François Bry and Sebastian Schaffert

Institute for Computer Science, University of Munich, Germany
corresponding address: schaffer@informatik.uni-muenchen.de

Abstract. Reasoning on the Web is receiving an increasing attention because of emerging fields such as Web adaption and Semantic Web. Indeed, the advanced functionalities striven for in these fields call for reasoning capabilities. Reasoning on the Web, however, is usually done using existing techniques rarely fitting the Web. As a consequence, additional data processing like data conversion from Web formats (e.g. XML or HTML) into some other formats (e.g. classical logic terms and formulas) is often needed and aspects of the Web (e.g. its inherent inconsistency) are neglected. This article first gives requirements for an entailment tuned to reasoning on the Web. Then, it describes how classical logic's entailment can be modified so as to enforce these requirements. Finally, it discusses how the proposed entailment can be used in applying logic programming to reasoning on the Web.

1 Introduction

Emerging Web applications and issues primarily call for *reasoning* capabilities. E.g. Semantic Web applications rely on declarative meta-data specifying the content of Web pages and Web sites and on description logic and/or ontology reasoning for processing these meta-data. Web services similarly rely on declarative meta-data for describing software resources made available on the Web.

There exist languages for expressing such meta-data (e.g. RDF, OWL [1,2]) and also systems for reasoning on meta-data expressed in such languages (e.g. Triple, FaCT [3,4]). However, such languages and/or reasoning systems are restricted to working with meta-data in a specific format and are not capable of retrieving and reasoning with *any* kind of data on the Web, i.e. Web pages and databases as well as – but not limited to – semantic annotations and ontological data. This article investigates a language capable of this.

Such reasoning languages have a multitude of applications. For instance, contract negotiation often takes place when Web services are traded. Wrappers are tools for a structure aware data retrieval that often rely on declarative structure specifications such as grammars, DTD [5] and/or schemas [6]. Data mediators convert data structured according to one DTD or schema into data after another DTD or schema. Adaptive Web systems provide and/or render Web data depending on contexts specifying e.g. user preferences or rendering device characteristics. The adaption of data to context is often realized with rule-based expert systems, i.e. a form of reasoning systems.

Reasoning on Web resources calls for methods better fitting the Web context than classical logic and conventional logic programming and automated reasoning. With conventional methods, additional data processing like data conversion from Web formats (e.g. XML [5] or HTML [7]) into some other formats (e.g. classical logic terms and formulas) is often needed and aspects of the Web (e.g. the need for partial queries on graph-shaped terms and the Web's inherent inconsistency) are neglected.

This article first gives a few requirements for an entailment tuned to Web applications. Instead of tree-shaped terms, terms possibly containing cyclic references are needed for such references often occur in XML [5] and HTML [7] data. Atomic formulas are needed using which it might be possible to express partial queries like with XPath [8] or XQuery [9]. A notion of formula satisfaction recursively defined on the formulas' structure is desirable since reasoning on the Web should be kept as "local" as possible. Also, meta-level, inconsistency tolerant, and nonmonotonic forms of reasoning are needed for Web applications. This article defines a notion of an entailment relation fulfilling these requirements. Finally, it discusses how this entailment relation can be used in applying logic programming to reasoning on the Web.

2 Semistructured Expressions

In contrast to the tuples of a relational database that are tree shaped, Web pages correspond to nested, possibly cyclic graphs. Abstracting from XML [5] and HTML [7], Web pages are conveniently formalised as "semistructured data" [10]. Semistructured data items are conveniently represented by "semistructured expressions" (short "sse") [10] that are defined by the following grammar:

```
                <sse> := ( oid "@" )? ( label | label <list>) .
               <list> := <ordered-list> | <unordered-list> .
   <ordered-list> := "[" <sse-str-ref> ( "," <sse-str-ref> )* "]" .
 <unordered-list> := "{" <sse-str-ref> ( "," <sse-str-ref> )* "}" .
    <sse-str-ref> := <sse> | """ string """ | "^" oid .
```

In this grammar, expressions between < and > are non-terminal symbols. "@", "[" , "{", "]", "}", """, and "^" are terminal symbols. "string", "oid" and "label" denote strings, tags and object identifiers, respectively. As usual in the formalisation of programming languages, these three symbols are terminal symbols to which values are assigned.

Slightly extending over XML and HTML, parentheses [ ] are used for expressing a compulsory order (e.g. f[a, b] and f[b, a] denote different data items), while parentheses { } serve to express that the order is irrelevant (e.g. both f{a, b} and f{b, a} denote the same data item). Ordered subterms are useful in representing (structured) texts and unordered subterms are useful in representing (structured) database items.

Semistructured expressions give rise to expressing tuples. E.g.

```
flight { number             [ "AF123"  ],
         departure-time     [ "1230" ],
         arrival-time       [ "1405" ],
         departure-airport  [ "Munich" ],
         arrival-airport    [ "Paris" ] }
```

is a possible representation of a tuple from a relation "flight" with attributes "number", "departure-time", "arrival-time", "departure-airport", and "arrival-airport". Semistructured expressions also give rise to representing cyclic XML documents like the following (where "key" is an attribute of type ID and "person-ref" is an attribute of type IDREF):

```
<persons>
     <person key="bry">
       <vorname>François</vorname>
       <nachname>Bry</nachname>
       <friend person-ref="schaffert"/>
     </person>
     <person key="schaffert" >
       <vorname>Sebastian</vorname>
       <nickname>Wastl</nickname>
       <nachname>Schaffert</nachname>
       <friend person-ref="bry"/>
       <friend person-ref="schaffert"/>
     </person>
</persons>
```

This XML document can be expressed as a semistructured expression as follows:

```
persons [ &1 @ person [ first-name [ "François" ],
                        last-name [ "Bry" ],
                        friend [ ^ &2 ]  ],
          &2 @ person [ first-name [ "Sebastian" ],
                        nickname [ "Wastl" ],
                        last-name [ "Schaffert" ],
                        friend [ ^ &1 ],
                        friend [ ^ &2 ]  ]  ]
```

Note that this semistructured expression contains two (directed) cycles.

Object identifiers in a semistructured expression are assumed to fulfil the following well-formedness conditions:

- Every object identifier &n referred to (i.e. occurring right of ^) in a semistructured expression, is also defined (i.e. occurs left of @) in this semistructured expression.
- An object identifier &n is defined (i.e. occurs left of @) at most once in a semistructured expression.

It might be convenient to also require that an object identifier &n defined in a semistructured expression is also referred to in this semistructured expression.

Without loss of generality, some features of XML and HTML (such as attributes, namespaces, DTD and/or schemas) that are not relevant to the present study and redundancies of XML and HTML (such as the three referencing formalisms through

ID-IDREF attributes, URIs, and Links) are not conveyed in semistructured expressions. Note that attributes can be expressed as semistructured expressions using an unordered collection of strings.

In the literature, slightly different formalisms are used for semistructured expressions. E.g. in [10] unordered children are not considered, one-element lists are represented without parentheses (e.g. instead of "first-name["François"]", [10] writes "first-name:"François""), object identifier definitions and object identifier references both appear to the right of an expression.

3 Requirements on a Logic for Reasoning on the Web

The paradigm considered in the following is to view a Web page as a ground atom and the Web as a (very large) set of Web pages. This paradigm translates into logic the view of a Web page as a data item and of the Web as a (large and highly distributed) database which is common in database research. A logic fulfilling the following requirements would make it possible to express Web-related reasoning problems directly, i.e. without translation of syntax and/or reasoning paradigms.

Terms Shaped as Rooted Graphs. Since Web pages (and therefore semistructured data) are shaped as rooted graphs possibly including cycles, a logical language with terms similarly shaped would ease reasoning with Web data.

Terms as Formulas. There is no natural way to classify Web page (and therefore semistructured) constructs into constructs interpreted as predicate and constructs interpreted as function symbols. Thus, there is no natural way to classify the subexpressions of a Web page (or semistructured expression) into terms and formulas. Therefore, a logical language not distinguishing between terms and formulas would be natural for reasoning on the Web.

Also, the *resource description framework* (RDF) proposed by the W3C [1] as a means to add semantics to web pages does not make such a distinction. In RDF, statements can both be interpreted as predicates and as objects.

Partial Queries. Queries on the Web as expressed in XQuery [9] and XPath [8] might specify Web pages partially. Therefore, a logical language for reasoning on the Web should make partial queries possible.

Satisfaction Recursively Defined. Classical logic satisfaction is defined by structural recursion: The truth value of a formula in an interpretation is defined in terms of the truth value of its subformulas in this interpretation. Recursive definitions of formula satisfaction give rise to evaluation procedures that are *local* to the formulas to evaluate and therefore can be efficiently processed against large data and/or knowledge bases. Therefore, a formula satisfaction recursively defined on the formulas' structures is desirable.

Meta-level Statements. Reasoning with meta-level assertions like database integrity constraints (i.e. statements that some Web data or sites should, or could, but not necessarily do fulfil) is desirable for advanced (e.g. adaptive or semantic Web) applications.

Inconsistency Tolerance. Because the Web is inherently heterogeneous, it is inherently inconsistent. Therefore, an inconsistency tolerant entailment relation is desirable for reasoning on the Web.

Nonmonotonic Negation. Nonmonotonic negation is needed on the Web as it is needed in relational databases: I.e. missing statements (e.g. a missing flight) allow to conclude the truth of the statement's negation (e.g. a missing flight does not exist).

4 Formulas for Reasoning on the Web

Building upon semistructured expressions (cf. above Section 2), a logical language is proposed for reasoning on the Web. In this language, semistructured expressions are a special kind of ground atomic formulas. Atomic formulas extend semistructured expressions with logical variables and a "descendant" construct used for "partial queries".

4.1 Atomic Formulas

Atomic formulas extend semistructured expressions with (logical) variables and with the descendant construct *desc*. These constructs can be informally understood as follows: Variables range over semistructured expressions and the descendant construct *desc* gives rise to specify a subexpression at any depth. Atomic formulas (or atoms) are defined by the following grammar:

```
          <atom> := ( oid "@" )? ( "desc" )?
                    (variable | label | ( variable | label ) <list>) .
          <list> := <ordered-list> | <unordered-list> .
  <ordered-list> := "[" <atom-str-ref> ( "," <atom-str-ref> )* "]" .
<unordered-list> := "{" <atom-str-ref> ( "," <atom-str-ref> )* "}" .
  <atom-str-ref> := <atom> | """ string """ | "^" oid .
```

Note that variables can occur everywhere except at the place of an object identifier. As in Prolog, identifiers beginning with an upper case letter will denote variables. The following atomic formula can be used for querying a Web page listing flights like an example of Section 2:

```
flight {
   number             [ Nb ],
   departure-time     [ Time1 ],
   arrival-time       [ Time2 ],
   departure-airport  [ "Munich" ],
   arrival-airport    [ "Paris" ]
}
```

The following atomic formula is a *partial query* to the same Web page returning the numbers (as bindings for the variable Nb) of the flights to Paris:

```
flight {
   number             [ Nb ],
   arrival-airport    [ "Paris" ]
}
```

Note the difference from classical logic and Prolog that require such a query to explicitely mention all the attributes of a flight tuple.

The descendant construct is a further means for expressing *partial queries*. The following three atomic formulas are partial queries to the persons Web page of Section 2. The first query evaluate to false. The second query evaluates to true returning the binding last-name/Label. The third evaluates to true (because of the references between the 'person' expressions).

```
desc "Mary"
persons { desc Label [ "Schaffert" ] }
desc person [ desc "Sebastian", desc "François" ]
```

Ordered lists in queries are satisfied only by ordered lists in semistructured expressions, while unordered lists in queries are satisfied by both ordered and unordered lists in semistructured expressions. Thus, evaluated against $f[g[a,b],g\{c,d\}]$ the query *desc* $g[X,Y]$ returns the single answer $a/X,b/Y$ while *desc* $g\{X,Y\}$ returns the two answers $a/X,b/Y$ and $c/X,d/Y$. Unordered lists are not easily expressible in a DTD. A similar notion is achieved with *all* groupings of XML Schema [6].

4.2 Compound Formulas

Compound formulas are built up as usual from atomic formulas using the connectives $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\neg$ and the quantifiers $\forall$ and $\exists$. E.g. the following is an open formula specifying one-stop connections from Munich to London:

```
flight {departure-airport ["Munich"], arrival-airport [Via]} ∧
   flight {departure-airport [Via], arrival-airport ["London"] }
```

The following is a closed formula expressing that every flight from A to B has a return flight:

```
∀ Nb1 ∀ D ∀ A
flight{number[Nb1], departure-airport[D], arrival-airport[A]}
  ⇒ ∃ Nb2
  flight{number[Nb2], departure-airport[A], arrival-airport[D]}
```

4.3 Propositional Formulas

As defined in 2, a semistructured expression, might be a label only (e.g. `a` or `flight`). Such semistructured expressions correspond to XML and HTML empty elements [5,7]. They might be seen as classical logic propositional variables or 0-ary predicate symbols, i.e. atomic formulas of propositional logic. Let us call such semistructured expressions "propositional atoms". Formulas build up (as defined in Section 4.2) from propositional atoms have exactly the form of propositional logic formulas. In the following, they are called "propositional formulas".

5 Entailment: Positive Formulas

The meaning of positive formulas (i.e. formulas in which no negations explicitely occur) has been informally introduced in Section 4. It is now formalised.

5.1 Interpretations

In a first approximation, an interpretation is conveniently defined as a set of semistructured expressions. Since semistructured expressions represent Web pages, this definition is well suited to reasoning on the Web. This definition is refined below in Section 6.1 so as to accommodate negation as discussed in Section 3.

The notion of interpretation considered here and in Section 6.1 can be further developed assigning to every label (or tag) a "multi-arity relation" (i.e. a subset of $\bigcup_{n \in N} \mathcal{E}^n$ where $\mathcal{E}$ is the set of semistructured expressions). This leads to an novel and interesting notion of relation composition. For space reasons, this is not further developed here.

The definition of interpretations considered here might be unusual from the angle of classical logic. Arguably, it is rather natural from the angle of logic programming since it is quite close to Herbrand interpretations.

Because queries on the Web might be partial specifications, some subexpressions of a semistructured expression true in an interpretation $\mathcal{I}$ are also true in $\mathcal{I}$. Consider for example the following (singleton) interpretation $\mathcal{F}$:

```
flights{
          flight {
             number             [ AF1 ],
             departure-time     [ 1230 ],
             arrival-time       [ 1405 ],
             departure-airport  [ "Munich" ],
             arrival-airport    [ "Paris" ]
          },
          flight {
             number             [ AF2 ],
             departure-time     [ 1530 ],
             arrival-time       [ 1615 ],
             departure-airport  [ "Paris" ],
             arrival-airport    [ "Munich" ]
          }
}
```

So as to accommodate partial queries as described in Section 4.1, the following semistructured expressions (among others) must be satisfied (or true) in $\mathcal{F}$:

```
flights
flights { flight }
flights { flight { number } }
flights { flight { number [ AF1 ] } }
flights { flight { number [ AF1 ] }, flight { number [ AF2 ] } }
```

5.2 Rooted Simulation

A notion of rooted simulation is used below in Section 5.3 in formalising the satisfaction of semistructured expressions and atomic formulas in an interpretation. The following definition is inspired from [11,12] and refines the simulation considered in [13]. Recall that a (directed) rooted graph $G = (V, E, r)$ consists in a set V of vertices, a set E of edges (i.e. ordered pairs of vertices), and a vertex r called the root of G such that there is in G a path from r to each vertex of G.

Definition 1 (Rooted Graph Simulation). *Let $G_1 = (V_1, E_1, r_1)$ and $G_2 = (V_2, E_2, r_2)$ be two rooted graphs and let $\preceq \subseteq V_1 \times V_2$ be an order relation. A relation $\mathcal{S} \subseteq V_1 \times V_2$ is a* rooted simulation *of G_1 in G_2 with respect to $\preceq$ if:*

1. *$r_1 \; \mathcal{S} \; r_2$.*
2. *If $v_1 \; \mathcal{S} \; v_2$, then $v_1 \preceq v_2$.*
3. *If $v_1 \; \mathcal{S} \; v_2$ and $(v_1, v_1') \in E_1$, then there exists $v_2' \in V_2$ such that $v_1' \; \mathcal{S} \; v_2'$ and $(v_2, v_2') \in E_2$*

A rooted simulation $\mathcal{S}$ of G_1 in G_2 with respect to $\preceq$ is minimal *if there are no rooted simulations $\mathcal{S}'$ of G_1 in G_2 with respect to $\preceq$ such that $\mathcal{S}' \subset \mathcal{S}$ (and $S \neq S'$).*

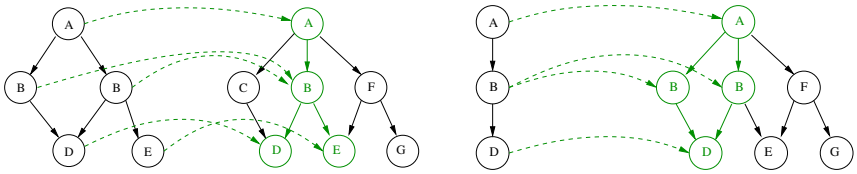

Fig. 1. Rooted Graph Simulations (with respect to vertex adornment equality)

Definition 1 does not preclude that two distinct vertices v_1 and v_1' of G_1 are simulated by the same vertex v_2 of G_2, i.e. $v_1 \; \mathcal{S} \; v_2$ and $v_1' \; \mathcal{S} \; v_2$. Figure 1 gives examples of simulations with respect to the equality of vertex adornments. The simulation of the right example is not minimal.

In the following, the order relation $\preceq$ considered is "equality of labels" and "compatibility of grouping", i.e. if v_1 has label l_1 and unordered children and if v_2 has label l_2 and ordered children, then $v_2 \not\preceq v_1$ even if $l_1 = l_2$ and $v_1 \preceq v_2$ if $l_1 = l_2$.

A semistructured expression E induces as follows a graph G_E whose vertices are adorned with labels of E and their "kinds of grouping", i.e. ordered ([ ]) or unordered ({ }): The vertices of G_E are those subexpressions of E that are semistructured expressions, a vertex of G_E is adorned with the leftmost label and kind of grouping of the semistructured expression it corresponds to, G_E has an edge (E_1, E_2) if E_2 is an immediate, proper subexpression of E_1 or if an immediate, proper subexpression of E_1 is a reference to E_2. Figure 2 illustrates this view of semistructured expressions as graphs.

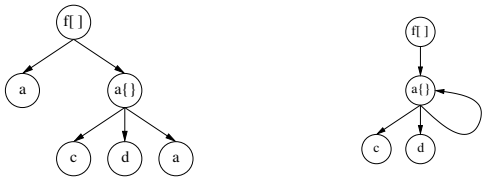

Fig. 2. $f[a, a[c, d, a]]$ and $f[\&1 \; @ \; a\{c, d, \hat{}\&1\}]$ as graphs

Considering the view of semistructured expressions as graphs, the notion of rooted simulation immediately extends to semistructured expressions. Intuitively, there exists a simulation of a semistructured expression E_1 in a semistructured expression E_2 if the labels and the structure of E_1 can be found in E_2 (cf. Figure 3).

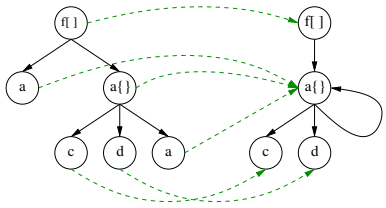

Fig. 3. Minimal simulation of $f[a, a\{c, d, a\}]$ in $f[\&1 \ @ \ a\{c, d, \hat{} \ \ \&1\}]$

5.3 Satisfaction of Atomic Formulas

Recall that an interpretation is a set of semistructured expressions. A semistructured expression E is satisfied (i.e. true) in an interpretation $\mathcal{I}$ if for some semistructured expression $E' \in \mathcal{I}$ there exists a minimal rooted simulation of E in E'. In particular, E is satisfied in $\mathcal{I}$ if $E \in \mathcal{I}$ (since vertex identity is a rooted simulation of E in itself). Thus, interpretations must be *closed under rooted simulation:* if $\mathcal{I}$ is an interpretation, $E \in \mathcal{I}$, and E' is a semistructured expression simulated in E, then $E' \in \mathcal{I}$.

This definition conveys the notion of partial queries to those ground atomic formulas that are semistructured expressions. It is extended to atomic formulas with variables or descendant constructs by extending the notion of rooted simulation of Section 5.2 as follows (cf. Figure 4 for an illustration):

Definition 2 (Formula Satisfaction – Part 1).

- *There exists a minimal rooted simulation of an atomic formula desc A in a semistructured expression E if there exists a subexpression E' of E and a minimal rooted simulation of A in E'.*
- *There exists a minimal rooted simulation of an atomic formula A in a semistructured expression E if there exists a grounding substitution of A (i.e. an assignment of semistructured expressions for variables in A) and a minimal rooted simulation of $A\sigma$ in E.*

5.4 Satisfaction of Compound Formulas

The satisfaction of compound formulas in which no negations explicitely occur is defined as usual recursively on the formulas' structures. The satisfaction of formulas with explicit negation is handled in a nonstandard manner in Section 6.1 below.

Definition 3 (Formula Satisfaction – Part 2). *Let $\mathcal{I}$ be an interpretation. Let F, F_1, and F_2 be formulas. Let X be a variable.*

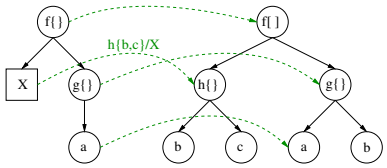

Fig. 4. Minimal simulation of the atomic formula $f\{X, g\{a\}\}$ in the semistructured expression $f[h\{b, c\}, g\{a, b\}]$

- $F_1 \wedge F_2$ *is satisfied in* $\mathcal{I}$ *if both* F_1 *and* F_2 *are satisfied in* $\mathcal{I}$.
- $F_1 \vee F_2$ *is satisfied in* $\mathcal{I}$ *if at least one of* F_1 *and* F_2 *is satisfied in* $\mathcal{I}$.
- $F_1 \Rightarrow F_2$ *is satisfied in* $\mathcal{I}$ *if* $\neg F_1 \vee F_2$ *is satisfied in* $\mathcal{I}$.
- $F_1 \Leftrightarrow F_2$ *is satisfied in* $\mathcal{I}$ *if both* $\neg F_1 \vee F_2$ *and* $F_1 \vee \neg F_2$ *are satisfied in* $\mathcal{I}$.
- $\forall X F$ *is satisfied in* $\mathcal{I}$ *if for all semistructured expressions* E $F[E/X]$ *is satisfied in* $\mathcal{I}$.
- $\exists X F$ *is satisfied in* $\mathcal{I}$ *if for some semistructured expression* E $F[E/X]$ *is satisfied in* $\mathcal{I}$.

5.5 Entailment

Entailment between formulas or sets of formulas F and G is defined as usual: $F \models G$ if every interpretation satisfying F also satisfies G. A *(paraconsistent) model* of a set $\mathcal{F}$ of formulas is a (paraconsistent) interpretation satisfying each formula in $\mathcal{F}$. A model $\mathcal{M}$ of $\mathcal{F}$ is *minimal* if no strict subsets of $\mathcal{M}$ are (paraconsistent) models of F.

5.6 Satisfaction of Propositional Formulas

On propositional atoms, i.e. atomic formulas reduced to labels (cf. Section 4.3), Definition 2 coincides with classical logic satisfaction. Therefore, the satisfaction of a propositional positive formula resulting from Definitions 2 and 3 coincides with that of classical logic.

6 Entailment: Integrity Constraints and Nonmonotonic Reasoning

In this section, the notion of interpretation introduced in Section 5.1 is refined yielding a framework for defining the satisfaction of negated formulas in a (nonstandard) manner.

6.1 Paraconsistent Interpretations

In the (paraconsistent) interpretations defined below both a negated formula $\neg F$ and its negation $\neg\neg F$ might be satisfied. In other words, classical logic's double negation elimination (i.e. $\neg\neg F \models F$) is dropped. Keeping with relational databases and logic programming, if an atomic formula A is true (false, resp.) in an interpretation $\mathcal{I}$, then $\neg A$ is false (true, resp.) in $\mathcal{I}$, and if $\neg\neg A$ is true (false, resp.) in $\mathcal{I}$, then $\neg\neg\neg A$ is false

(true, resp.) in $\mathcal{I}$. Furthermore, if a (paraconsistent) interpretation satisfies a formula F, then $\mathcal{I}$ also satisfies $\neg\neg F$. Thus, an atomic formula A can be interpreted as follows (the first and last interpretations are like in classical logic):

A	$\neg A$	$\neg\neg A$	$\neg\neg\neg A$
true	false	true	false
false	true	true	false
false	true	false	true

Definition 4 (Interpretation). *A (paraconsistent) interpretation $\mathcal{I}$ is a set of semistructured expressions or doubly negated semistructured expressions such that:*

1. *$\mathcal{I}$ is closed under rooted simulation, i.e. if $E \in \mathcal{I}$ and if E' is a semistructured expression simulated in E, then $E' \in \mathcal{I}$.*
2. *If E is a semistructured expression and if $E \in \mathcal{I}$, then $\neg\neg E \in \mathcal{I}$.*

A (paraconsistent) interpretation $\mathcal{I}$ is consistent *if for all $\neg\neg E \in \mathcal{I}$, $E \in \mathcal{I}$. It is* inconsistent *otherwise.*

Dropping the elimination of double negation avoids problems found in classical logic. Consider for example a course database with integrity constraints as follows:

$$\begin{array}{ll} \text{DB:} & course[\text{"}CS1\text{"}] \\ & course[\text{"}CS2\text{"}] \\ & teaches[\text{"}Anna\text{"}, \text{"}CS1\text{"}] \\ \text{IC:} & \forall X.course[X] \;\Rightarrow\; \exists Y.teaches[Y, X] \end{array}$$

In classical logic, for which a so-called "open world assumption" holds, there exists a model for this database and integrity constraint (since it is not explicitly stated that there is no teacher for CS1), and thus it is consistent. Database systems avoid this problem by relying on a so-called "closed world assumption" or Clark's completion [14] and by distinguishing between basic data (like course["CS1"]) and integrity constraints. Such a distinction is not necessary with paraconsistent interpretatrions as above because integraity constraints are prefixed with a double negation. The integrity constraint of the example above is thus expressed as:

$$\text{IC:} \neg\neg \left(\forall X.course[X] \;\Rightarrow\; \exists Y.teaches[Y, X]\right)$$

A paraconsistent model satisfying $course[\text{"}CS1\text{"}]$, $course[\text{"}CS2\text{"}]$, and the formula above does not have to satisfy $teaches[a, \text{"}CS2\text{"}]$ for some a. Instead, it suffices that $\neg\neg teaches[a, \text{"}CS2\text{"}]$ holds for some a, intuitively expressing that $teaches[a, \text{"}CS2\text{"}]$ *should* hold for some a.

6.2 Satisfaction of Negated Formulas

The satisfaction of negated formulas in a (paraconsistent) interpretation $\mathcal{I}$ is defined recursively on the formulas' structure by extending Definition 2 of Section 5.4 with the following rules:

Definition 5 (Formula Satisfaction – Part 3). *Let $\mathcal{I}$ be an interpretation. Let A be an atomic formula. Let F be a formula.*

– $\neg A$ *is satisfied in* $\mathcal{I}$ *if* A *is not satisfied in* $\mathcal{I}$*, i.e. there are no semistructured expressions* $B \in \mathcal{I}$ *with a minimal rooted simulation of* A *in* B*.*
– $\neg\neg\neg A$ *is satisfied in* $\mathcal{I}$ *if there are no* $\neg\neg B \in \mathcal{I}$ *with a minimal rooted simulation of* A *in* B*.*
– $\neg\neg\neg\neg F$ *is satisfied in* $\mathcal{I}$ *if* $\neg\neg F$ *is satisfied in* $\mathcal{I}$*.*

In a word, four and more than four nested negations are treated in the spirit of classical logic, double negations are not eliminated, and $F \models \neg\neg F$. Two kinds of positive literals are available, atoms and doubly negated atoms. Negated atoms and doubly negated atoms are treated in the relational database and logic programming style. From Definitions 4, 2, 3, and 5, it (easily) follows that for all formulas F, $F \models \neg\neg F$ (but $\neg\neg F \not\models F$).

Note that a consistent interpretation of propositional formulas induces a classical logic interpretations of these formulas. This is not the case of an inconsistent interpretation in which for some A, $\neg\neg A$ is true but A is not true.

Section 6.3 and Section 6.4 below point to the advantages of this unusual treatment of negation for reasoning on the Web.

6.3 Meta-level Reasoning: Integrity Constraints

Positive and general program clauses and programs are defined as usual but referring to the nonstandard atomic formulas defined in Section 4.1:

Definition 6 (Program Clauses and Programs).

– A program clause *is an expression of the form* $A \leftarrow B_1, \ldots B_n$ *where the* B_i *are atomic formulas or negated atomic formulas and* A *is an atomic formula in which no desc constructs occur. It denotes the formula* $\forall X_1 \ldots \forall X_m\ (B_1 \wedge \ldots \wedge B_n) \Rightarrow A$ *where* $X_1, \ldots, X_m$ *are the variables occurring in* $B_1, \ldots, B_n$*, or* A*. If all* B_i *are atomic formulas, then it is a* positive clause*, else a* general clause*.*
– A positive (general, resp.) program *is a finite set of positive (general, resp.) program clauses. A* propositional program *is a program in the clauses of which only propositional atomic formulas or propositional negated atomic formulas occur.*

Precluding *desc* constructs in clauses' heads ensure that positive programs have only one minimal model. Indeed, a clause like $a\{desc\, b\} \leftarrow B_1, \ldots B_n$ defines infinitely many atoms e.g. $a\{b\}$, $a\{a\{b\}\}$, $a\{a\{b\}, a\{b\}\}$, etc.

Integrity constraints to a program P are closed formulas that might (or, depending on the application, should, or could) logically follow from P. If C is a set of integrity constraints to a program P, the problem called "integrity verification" is to decide whether $P \models C$. If this is the case, P is said to satisfy the integrity constraints, otherwise to violate them. Note that the data specified by P, i.e. the model(s) of P, should not depend on C. Thus, integrity constraints are statements on P, i.e. meta-level statements.

If integrity constraints are represented as doubly negated (closed) formulas $\neg\neg F$, then the entailment relation of Section 6.1 suffices to solving integrity verification problems:

Proposition 1. *A positive program* P *satisfies a set of integrity constraints* C *represented as doubly negated formulas if and only if the minimal (paraconsistent) models of* $P \cup C$ *are consistent.*

Proof. Let P be a positive program and let C be a set of integrity constraints (of the form $\neg\neg F$) to P. Let $\mathcal{I}$ be a paraconsistent interpretation such that $\mathcal{I} \models P \cup C$, i.e. a paraconsistent model of $P \cup C$. Clearly, the subset of $\mathcal{I}$ consisting in all atoms of $\mathcal{I}$ is a standard Herbrand model of P (as defined in [15]). If $\mathcal{I}$ is minimal, then the subset of $\mathcal{I}$ consisting in all atoms in $\mathcal{I}$ is the minimal Herbrand model of P (in the standard sense). Now, observe that $P \models C$ if and only if $P \cup C$ has no minimal models that are inconsistent (i.e. models $\mathcal{M}$ with $\neg\neg A \in \mathcal{M}$ and $A \notin \mathcal{M}$ for some semistructured expression A).

6.4 Nonmonotonic Reasoning

With the entailment relation of Section 6.1, a ground program clause $A \leftarrow B_1, \ldots B_n, \neg C_1 \vee \ldots \vee \neg C_m$ is logically equivalent to $A \vee \neg B_1 \vee \ldots \vee \neg B_n \vee \neg\neg C_1 \vee \ldots \neg\neg C_m$. It is not only satisfied in interpretations in which A, or some $\neg B_i$, or some C_j is true, but also in interpretations in which some $\neg\neg C_j$ is true. Among such (paraconsistent) interpretations are inconsistent interpretations satisfying $\neg\neg C_j$ but not satisfying C_j. Such interpretations "gives room" for interpreting so-called cycles of recursion through negation with an odd length [16] in a quite natural manner. E.g. the clause $p \leftarrow \neg p$ has a single minimal (inconsistent) paraconsistent model: $\{\neg\neg p\}$. Major advantages of the treatment of negation proposed above is that it extends the Stable Model Semantics [17] and gives it a "minimal model setting":

Proposition 2. *Let P be a propositional program. Every consistent model of P (in the sense of Section 6.1) is stable. Every stable model of P (in the sense of [17]) characterises a consistent model of P (in the sense of Section 6.1).*

If $\mathcal{M} = \{A_1, \ldots, A_k\}$ is a stable model, then the model in the sense of Section 6.1 $\mathcal{M}$ is said to characterise is $\{A_1, \neg\neg A_1, \ldots, A_k, \neg\neg A_k\}$.

Proof. Proposition 2 follows from the characterisation of minimal (paraconsistent) models given below in Proposition 3 which rephrases the program transformation of [17] and extend it to general formulas.

Proposition 3. *Let $\mathcal{M}$ be a (paraconsistent) model of a set of formulas S. Let $\overline{\mathcal{M}} = \{\neg E \mid E$ semistructured expression and $E \notin \mathcal{M}\} \cup \{\neg\neg\neg E \mid L$ semistructured expression and $\neg\neg E \notin \mathcal{M}\}$. $\mathcal{M}$ is a minimal model of S if and only if for all $K \in \mathcal{M}$, $S \cup \overline{\mathcal{M}} \models K$.*

Proof. **Necessary condition.** Assume $\mathcal{M}$ is a minimal (paraconsistent) model of S. If $\mathcal{M} = \emptyset$, then the property trivially holds. Otherwise, let $K \in \mathcal{M}$. If $S \cup \overline{\mathcal{M}} \not\models K$, then by Definition 5 $S \cup \overline{\mathcal{M}} \cup \{\neg K\}$ has a minimal (paraconsistent) model $\mathcal{N}$. By definition of $\mathcal{N}$, $\mathcal{N} \models \neg K$, hence by Definition 5, $\mathcal{N} \not\models K$, i.e. $K \notin \mathcal{N}$. Since $\mathcal{N} \models \mathcal{S} \cup \overline{\mathcal{M}}$, $\mathcal{N} \subseteq \mathcal{M}$. Since $K \in \mathcal{M} \setminus \mathcal{N}$, $\mathcal{N} \neq \mathcal{M}$ contradicting that $\mathcal{M}$ is a minimal model of S. Thus, $S \cup \overline{\mathcal{M}} \models K$.
Sufficient condition. Assume that for all $K \in \mathcal{M}$, $S \cup \overline{\mathcal{M}} \models K$. If $\mathcal{M}$ is not a minimal (paraconsistent) model of S, then there exists a strict subset $\mathcal{N}$ of $\mathcal{M}$ such that $\mathcal{N} \models S$. Let $\overline{\mathcal{N}} = \{\neg E \mid E$ semistructured expression and $E \notin \mathcal{N}\} \cup \{\neg\neg\neg E \mid E$ semistructured expression and $\neg\neg E \notin \mathcal{N}\}$. From the necessary condition it follows that for all $K \in \mathcal{N}$, $S \cup \overline{\mathcal{N}} \models K$. Since $\mathcal{N}$ is a strict subset of $\mathcal{M}$ there exists $K \in \mathcal{M} \setminus \mathcal{N}$, hence $\neg K \in \overline{\mathcal{N}}$ and ($\star$) $S \cup \overline{\mathcal{N}} \models \neg K$. By hypothesis, $S \cup \overline{\mathcal{M}} \models K$. Since $\mathcal{N} \subset \mathcal{M}$, $\overline{\mathcal{M}} \subset \overline{\mathcal{N}}$. Therefore, ($\star\star$) $S \cup \overline{\mathcal{N}} \models K$. ($\star$) and ($\star\star$) are contradictory, refuting that $\mathcal{M}$ is not minimal.

7 Logic Programming for Reasoning on the Web

Many Web applications are based on (1) selecting data from the Web and (2) processing the selected data in manners that are particularly amenable to declarative programming, especially to logic programming. The entailment relation introduced in the previous sections has been conceived so as to support such applications that are briefly described below. It provides with the semantics of a prototype Web query language called Xcerpt [13,18].

Dynamic Web pages. Dynamic Web pages are texts or data that are dynamically generated when called. They make it possible for different pages to share data thus ensuring data consistency and to generate up-to-date Web pages from changeable data. Arguably, logic programming queries would be as convenient for the Web as views are in relational databases. Dynamic Web pages based on a logic query language would be more amenable to reasoning (e.g. for query optimisation purposes) than dynamic Web pages based on imperative scripts.

Adaptive Web. Most adaptive Web systems combine portions of text into Web pages depending on contexts specifying so-called user models (i.e. user preferences and/or rendering device characteristics) using rule-based systems. Arguably, a logic programming query language would be convenient to implement both, the retrieval of portions of text from the Web and their combination into context-dependent Web pages, thus considerably simplifying the implementation of adaptive Web systems.

Structure Transformations. Structure transformations are an essential application of languages such as XQuery [9] and XSLT [19]. Arguably, logic programming languages are especially convenient to express structure transformations because they are term (or pattern oriented) and because they are convenient to express recursion through term structures.

Styling. Styling, i.e. enriching an XML or HTML page with rendering parameters, is a special kind of transformation conveniently expressed in rule-based formalisms. CSS [20] is such a language that has interesting similarities with logic programming.

Semantically Aware Querying. Considering semantics annotations as expressed e.g. in ontologies while evaluating queries on the Web is an emerging research issue. To this aim, the reasoning system of ontologies is coupled with a programming language. Arguably, a logic programming query language could be used for both tasks.

8 Related Work

The work presented in this paper is related to query and transformation languages for XML [5] and semistructured data. XPath [8] and XQuery [9] are well known such languages. They are widely used Web standards. They are "navigational" in the sense that they express data retrieval in terms of root-to-node data item traversals, i.e. a rather procedural approach.

Other query languages for XML [5] and semistructured data do not build upon a navigational paradigm. UnQL [21] first proposed to express queries as terms (or

patterns) as in logic and in the present paper. Such query languages can be called "positional" because the relative positions of the data to retrieve, e.g. variables, are well conveyed in query terms (or patterns). A further positional query language is XMas [22]. Both UnQL and XMas are functional and inspired from the object database query language OQL [23]. Xcerpt (cf. `http://www.xcerpt.org`) is a further positional query and transformation language for XML and semistructured data. Xcerpt is based on the logic programming paradigm. The present paper is a contribution to Xcerpt's semantics. Xcerpt's operational semantics has been presented in [13].

Further query languages for XML and semistructured data are XSLT [19] and fxt [24]. XSLT is based on the matching of XML elements and on a built-in structural recursion of XML documents and it also offers a comprehensive collection of imperative programming constructs. A severe limitation of XSLT is that it has no real notion of procedure: An output of an XSLT subprogram cannot be further processed within the same program. fxt builds on tree grammars and an efficient matching of regular expressions (describing the children of a node) against semistructured expressions. fxt's processing is based on tree automata. In some projects, Prolog has been adapted to process and/or query XML and/or semistructured data, e.g. in [25,26,27]. Such Prolog extensions and/or adaptions are inspiring for query languages for XML [5] and semistructured data.

The approach to nonmonotonic reasoning described in the present paper is reminiscent of a widespread, often empirical approach consisting in "duplicating" every predicate p (cf. e.g. [16,28,29]). [30] describes in more detail this approach in the framework of classical logic. Proposition 3 is an adaptation to the nonstandard models of Definition 4 of a result given in [31] for classical logic.

9 Conclusion

This article has first given requirement for logics for reasoning on the Web as needed for emerging applications such as Semantic Web and adaptive Web systems. Then, it has defined an entailment relation for such a logic. A first salient aspect of this entailment relation is that it conveys a notion of partial queries reminding of the Web query languages XPath [8] and XQuery [9]. A second salient aspect of the entailment relation is that the satisfaction of a formula is recursively defined on the formulas' structures, like in classical logic and unlike most nonmonotonic logics. Arguably, such definitions of formula satisfaction give rise to efficient inference procedures local to the formulas considered, thus well-suited to the Web. A third salient aspect of the entailment relation proposed in this paper is that it extends the Stable Model Semantics [17] and gives it a "minimal model setting". Finally, the article has briefly discussed how the proposed entailment relation can be used in applying logic programming to reasoning on the Web.

The work reported about in this paper is part of a project aiming at defining a (prototype) logic programming language for reasoning on the Web called Xcerpt [13,18].

References

1. W3C: Resource Description Framework (RDF). (1999)
2. W3C: Web Ontology Language (OWL). (2003)

3. Decker, S.: TRIPLE – an RDF query, inference, and transformation language. website (2002) http://triple.semanticweb.org/.
4. Horrocks, I.: The FaCT System. website (1999) http://www.cs.man.ac.uk/~horrocks/FaCT/.
5. W3C: Extensible Markup Language (XML) 1.0, Second Edition. (2000)
6. W3C: XML Schema Part 0: Primer; Part 1: Structures, Part 2: Datatypes. (2001)
7. W3C: XHTML 1.0: The Extensible HyperText Markup Language. (2000)
8. W3C: XML Path Language (XPath). (1999)
9. W3C: XQuery: A Query Language for XML. (2001)
10. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufmann (2000)
11. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing Simulations on Finite and Infinite Graphs. Technical report, Cornell Univ. (1996)
12. Milner, R.: An Algebraic Definition of Simulation between Programs. Memo aim-142, Stanford Univ. (1971)
13. Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: Proc. Int. Conf. on Logic Programming. LNCS, Springer-Verlag (2002)
14. Clark, K.L.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press (1978) 293–322
15. Lloyd, J.: Foundations of Logic Programming. Springer-Verlag (1987)
16. Apt, K.R., Bol, R.: Logic Programming and Negation: A Survey. J. Logic Programming **9** (1994)
17. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Proc. Int. Conf. on Logic Programming. (1988) 1070–1080
18. Bry, F., Schaffert, S.: A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web. (2002) (invited article).
19. W3C: Extensible Stylesheet Language (XSL). (2000)
20. W3C: Cascading Style Sheets, level 2. (1998)
21. Buneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. VLDB Journal **9** (2000)
22. Baru, C., Ludöscher, B., Papakonstantinou, Y., Velikhov, P., Vianu, V.: Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. In: Proc. QL'98 – The Query Languages Workshop. (1998)
23. Alashqur, A.M., Su, S.Y.W., Lam., H.: OQL: A Query Language for Manipulating Object-Oriented Databases. In: Proc. Int. Conf. on Very Large Data Bases. (1989)
24. Berlea, A., Seidl, H.: fxt – A Transformation Language for XML Documents. J. of Computing and Information Technology (2001)
25. Seipel, D.: Processing XML-Documents in Prolog. In: Proc. Workshop Logische Programmierung. (2002)
26. Heumesser, B., Seipel, D., Güntzer, U.: Flexible Processing of XML-Based Mathematical Knowledge in a Prolog-Environment. In: Proc. Int. Conf. on Mathematical Knowledge Management. LNCS (2003)
27. May, W.: A Logic-Based Approach to XML Data Integration. (2001) Habilitation Thesis.
28. Inoue, K., Sakama, C.: A Fixpoint Characterization of Abductive Logic Programming. J. Logic Programming (1996) 107–136
29. Lifschitz, V., Pearce, D., Valverde, A.: Strongly Equivalent Logic Programs. ACM Trans. Computational Logic **2** (2001) 526–541

30. Bry, F.: An Almost Classical Logic for Logic Programming and Nonmonotonic Reasoning. In: Proc. Paraconsistent Computational Logic. (2002)
31. Niemelä, I.: A Tableau Calculus For Minimal Model Reasoning. In: Proc. Workshop on Theorem Proving with Analytic Tableaux and Related Methods. LNAI, Springer-Verlag (1996)