Temporal Patterns for Document Verification

Mirjana Jakšić and Burkhard Freitag

Department of Informatics and Mathematics, University of Passau
{Mirjana.Jaksic, Burkhard.Freitag}@uni-passau.de

Technical Report, Number MIP-0805
Department of Informatics and Mathematics
University of Passau, Germany
November 2008

Abstract

Digital text, in particular hypertext, can be represented as a temporal structure based on the concept of narrative paths. Using computation tree logic (CTL) as a formal basis, consistency constraints about the document can then be expressed as temporal properties. These in turn can be verified against the document model by model checking.

Unfortunately, the average user can not be assumed to be familiar with temporal logics. Therefore, as an approach to fill the gap between formality and usability, we present a novel user-friendly high-level approach to the specification of temporal properties supporting an incremental construction of commonly used consistency criteria for web documents.

1 Introduction

The concept of consistency is commonly applied to databases, programs, protocols, concurrent processes, and systems but can be naturally extended to digital documents. Various notions of consistency and a wide range of consistency checking methods have been studied in the field of digital documents.

In this report we address the problem of specifying consistency criteria for the purpose of verification of web documents. We focus on temporal properties of documents along standard reading paths. For example, we check whether in a web-based training (WBT) document *every description of a certain concept is followed by an example of the same concept.* This kind of consistency is particularly useful when having to ensure the document coherence and certain properties of the narrative flow, e.g. in e-learning or technical documentation.

The verification is performed by model-checking based on computation tree logic - CTL [4, 6]. Temporal logics are usually used for verification tasks in the application field of software engineering but there are also systems using temporal logic for hypertext verification [11, 12].

Applying a temporal logic such as CTL requires good mathematical knowledge and a lot of experience and usually involves considerable effort in terms of manpower and time. For this reason, a high-level mechanism supporting the process of formal specification is highly desirable. Our goal is to provide a user-friendly high-level specification scheme for temporal properties, which supports the incremental construction of commonly used consistency criteria for web documents.

Among the existing methods for high-level specification, pattern-based approaches are well established and widely used [3, 5, 9]. In many cases they do not require deep-level knowledge of the underlying specification formalism. We will show that specification patterns which originally have been introduced for the field of reactive systems [3] can be adapted and enhanced

for the purpose of specifying consistency properties of documents. Furthermore, we define an appropriate mapping of patterns onto CTL formulae. We also show that the construction of commonly used consistency conditions for web documents can be performed incrementally thus giving less experienced users the opportunity to proceed from low to higher complexity.
The contribution of this paper consists of:

- defining a set of specification patterns representing general temporal constraints that can be applied to express document consistency,
- showing how the proposed specification patterns can be used in the process of formalizing consistency criteria for web documents.

The paper is organized as follows. Section 2 describes the problem - addressed. Section 3 gives a brief introduction to computation tree logic and the temporal model of a web document. Section 4 introduces specification patterns for documents, section 5 deals with pattern transformation into CTL, while our specification tool is introduced in section 6. We conclude with a short summary (section 7).

2 Problem Description

Our aim is to check the consistency of the narrative structure of a document. We define the *narrative structure* of a document as a representation of the sensible orders of visiting, i.e., reading its fragments in standard situations (cf. [13]). The narrative structure of a web document is typically branching and not linear.
When browsing a document, the user, of course, does not have to follow its narrative structure, but in case she does, the presented content should make sense. Hence, the consistency criteria refer to all or some of the "recommended ways" of reading the document along its narrative structure. These "recommended ways" of reading the document are referred to as its *narrative paths* (see [12, 13]).
The narrative structure of a document is represented by a directed graph of *narrative units* (vertices) and *narrative relations* (edges).

Example 1 (Narrative structure)
Figure 1 depicts a fragment of a narrative structure of a web document taken from a web based training (WBT) about datastructures. In addition to the "main track" about datastructures, this fragment contains also an ex-course about abstract datatypes for advanced students.
The unit start is followed by the definition of datastructure. After this unit there are two possible branches to follow. The first one proceeds with an example of datastructure, then with a summary and a test about datastructure, and finally with the end unit. The other branch continues with

a definition of abstract datatypes, then subsequently with an example of abstract datatypes, afterwards with the summary and a test about datastructure, and finally with the end unit.

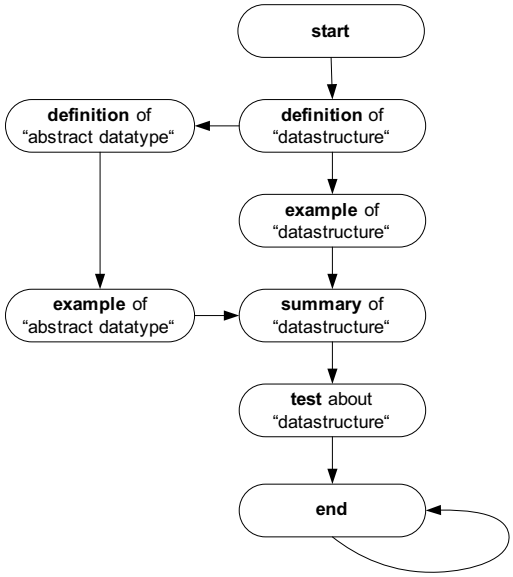

Figure 1: Narrative structure of a document

Let us consider the following sample consistency criteria:

1. *On all paths there exists a summary unit before the test unit.*
2. *Every definition of the topic datastructure is on all succeeding paths followed by an example of the same topic.*
3. *After the summary unit, no definition units are allowed.*

Obviously, criterion 1 holds in the structure of Figure 1. On both paths a summary unit exists immediately before the test unit.
On the other hand, criterion 2 does not hold in the given structure, because there is a path (start, definition of datastructure, definition of abstract datatypes, example of abstract datatypes, summary of datastructure, test about datastructure, end) with a definition of datastructure not being followed by an example of datastructure.
Finally, criterion 3 holds in the given structure, because both definitions (definition of datastructure, definition of abstract datatype) appear before the summary unit, concerning both possible paths. □

There are several possible ways to verify the consistency of web documents. XML documents can be verified using e.g. DTD, XSchema, or XPath-based

languages. In contrast to the narrative structure of web documents introduced above, which typically represents many sensible ways of reading, the XML data model assumes a linear order of document elements. Moreover, it can be shown that the representation of consistency criteria by means of a temporal logic is more compact as compared to a XML-based language (see e.g. [2]). Note also, that using a temporal logic is more general because the document format is not required to be XML.
Of course, temporal consistency criteria could be expressed using first order logic, too. However, [10] demonstrates that the representation of temporal conditions in first order logic leads to very complex expressions that are, in general, expensive to evaluate.
As compared to XML-based approaches and first order logic, temporal logic has several advantages, among them a compact representation of consistency criteria and a set of efficient verification procedures. In this paper, we use the temporal logic CTL to express consistency criteria affecting narrative paths.
The main steps of consistency specification and verification are depicted in Figure 2. Users appear in two different roles: First, there are document authors, who provide, organize, and maintain document fragments. Experienced authors may also be able to specify consistency criteria using the interface for pattern-based specification to be described later in this paper. Second, there are temporal logic experts who can specify complex criteria directly in CTL and maintain the verification model, if necessary.

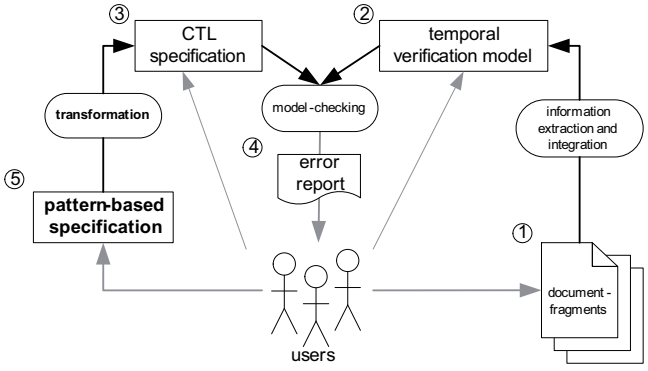

Figure 2: Automated verification of semi-structured documents

Assume that there are several text components, possibly in different formats (no. 1 in Figure 2). The information about the document's content and structure are available in the form of markup and external metadata or are provided by external information extraction tools. The collected information is represented by a *temporal verification model* (no. 2 in Figure 2) (see section 3) which essentially formalizes the narrative structure of the

document. This way an abstraction is provided from implementation details which are irrelevant for the verification tasks.
The specification criteria are expressed in CTL (no. 3 in Figure 2) and verified against the verification model by the CTL model checker. The verification results (counterexamples) are then presented to the user (no. 4 in Figure 2).
As a temporal logic, CTL is likely to be too demanding for non-expert users - which of course tend to be the majority - a user-level specification method based on specification patterns has been developed (no. 5 in Figure 2). Patterns represent commonly occurring requirements concerning the content and structure of documents (see Definition 10). Specification patterns are translated into CTL formulae.
Our approach to automated verification of semi-structured documentss is in detail presented in [14].

3 Computation Tree Logic

CTL [6] is a propositional, branching time, future tense temporal logic evaluated w.r.t. discrete points in time, so called states. CTL allows to distinguish properties, which hold on some path, from those that hold on all paths within a given structure. This property is useful for verification of non-linearly structured documents.
CTL defines a language over a countable set AP of application dependent atomic propositions and a finite, fixed set of connectives. Atomic propositions represent atomic statements, which can be either true or false at a given point in time. The set of connectives is defined by the syntax of CTL.

Definition 2 (CTL Syntax)
The set of CTL formulae is the minimal set of expressions, which are generated by the following grammar rule where a is a member of the set AP of application dependent atomic propositions:

$$p, q \longrightarrow \top \mid \bot \mid a \mid \neg p \mid p \wedge q \mid p \vee q \mid p \rightarrow q \mid$$
$$\mathsf{AX}\ p \mid \mathsf{EX}\ p \mid \mathsf{AF}\ p \mid \mathsf{EF}\ p \mid \mathsf{AG}\ p \mid \mathsf{EG}\ p \mid \mathsf{A}[p\ \mathsf{U}\ q] \mid \mathsf{E}[p\ \mathsf{U}\ q]$$

Each CTL temporal connective is a pair of symbols. The first part of the pair is a *path quantifier* - either A(all paths), or E(some path). The second one is a *temporal operator*: X(next p), F(eventually p), G(globally p), or U(p until q). □

Some examples of CTL formulae are:

1. *On some path eventually a test occurs.*

$$\mathsf{EF}\ test$$

2. *On all paths there is eventually a summary.*

$$\mathsf{AF}\ summary$$

3. *Start and end cannot hold at the same time.*

$$\mathsf{AG}\ \neg(start \wedge end)$$

4. *At any time, help is reachable within one step.*

$$\mathsf{AG}\ \mathsf{EX}\ help$$

5. *Whenever the user submits some data, it is confirmed in some next step.*

$$\mathsf{AG}\ (submit \rightarrow \mathsf{EX}\ confirm)$$

In these sample formulae, the expressions *test*, *summary*, *start*, *end*, *help*, *submit*, $confirm$ are propositional formulae expressing the property of being a test unit or summary unit and so on. CTL formulae are interpreted w.r.t. labelled state transition systems, i.e. CTL temporal structures, as defined below.

Definition 3 (CTL temporal structure)
A CTL temporal structure M is defined as a state transition system $M = (S, R, L)$ where:

- S is a nonempty set of states,
- $R \subseteq S \times S$ is a left-total binary transition relation,
- $L : S \mapsto \mathcal{P}(AP)$ is a labelling of states such that $L(s)$ is the set of atomic propositions that hold at state $s \in S$.

□

CTL temporal structures model processes in terms of states and state transitions. Properties of states are represented by sets of atomic propositions a, which are true at a given state.

Example 4 (Temporal structure of a document)
The temporal structure $M = (S, R, L)$ corresponding to the document, whose narrative structure is presented in Example 1 (Figure 1), is defined by

$$
\begin{aligned}
S &= \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\} \\
R &= \{(s_0, s_1), (s_1, s_2), (s_1, s_3), \\
&\quad (s_2, s_5), (s_3, s_4), (s_4, s_5), (s_5, s_6), (s_6, s_7), (s_7, s_7)\} \\
L &= \{s_0 \mapsto \{start\}, s_1 \mapsto \{definition, datastructure\}, \\
&\quad s_2 \mapsto \{example, datastructure\}, \\
&\quad s_3 \mapsto \{definition, abstract\_datatype\}, \\
&\quad s_4 \mapsto \{example, abstract\_datatype\}, \\
&\quad s_5 \mapsto \{summary, datastructure\}, \\
&\quad s_6 \mapsto \{test, datastructure\}, s_7 \mapsto \{end\}\}
\end{aligned}
$$

□

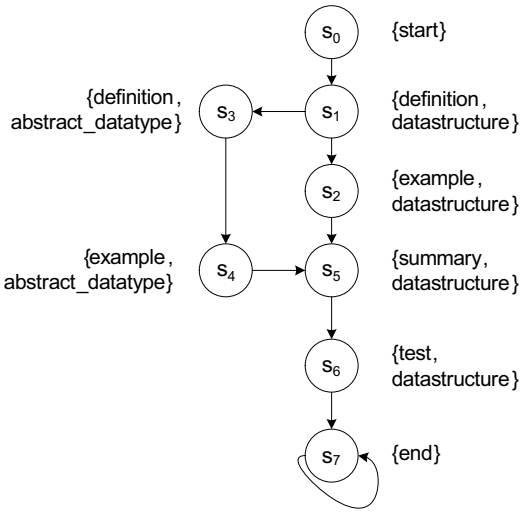

Figure 3: Temporal structure of a document

In the structure shown in Figure 3 one can distinguish two different paths, namely $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail s_5 \rightarrowtail s_6 \rightarrowtail s_7 \rightarrowtail ...$ (short for: $\{(s_0, s_1)$, (s_1, s_2), (s_2, s_5), (s_5, s_6), (s_6, s_7), $(s_7, s_7)\}$) and $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrowtail s_4 \rightarrowtail s_5 \rightarrowtail s_6 \rightarrowtail s_7 \rightarrowtail ...$.

Definition 5 (CTL Semantics)
Let $M = (S, R, L)$ be a temporal structure, and $s_0 \in S$ a state. Let $a \in AP$ be an atomic proposition and p, q CTL formulae.
The semantics of CTL defines when a CTL formula p is *true* in a structure $M = (S, R, L)$ at a state $s_0 \in S$, in symbols: $M, s_0 \models p$. The truth relation $\models$ is inductively defined as in [6]:

1. $M, s_0 \models \top$ and $M, s_0 \not\models \bot$.
2. $M, s_0 \models a$ iff $a \in L(s_0)$.
3. $M, s_0 \models \neg p$ iff $M, s_0 \not\models p$.
4. $M, s_0 \models p \wedge q$ iff $M, s_0 \models p$ and $M, s_0 \models q$.
5. $M, s_0 \models p \vee q$ iff $M, s_0 \models p$ or $M, s_0 \models q$.
6. $M, s_0 \models p \rightarrow q$ iff $M, s_0 \not\models p$ or $M, s_0 \models q$.
7. $M, s_0 \models \mathsf{AX}\ p$ iff for all s_1 such that $s_0 \rightarrowtail s_1$, we have $M, s_1 \models p$.
8. $M, s_0 \models \mathsf{EX}\ p$ iff for some s_1 such that $s_0 \rightarrowtail s_1$, we have $M, s_1 \models p$.
9. $M, s_0 \models \mathsf{AG}\ p$ iff for all paths $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail \ldots$, and for all s_i along the path, we have $M, s_i \models p$.
10. $M, s_0 \models \mathsf{EG}\ p$ iff there is a path $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail \ldots$ where for all s_i along the path, we have $M, s_i \models p$.
11. $M, s_0 \models \mathsf{AF}\ p$ iff for all paths $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail \ldots$ there is some s_i such that $M, s_i \models p$.
12. $M, s_0 \models \mathsf{EF}\ p$ iff there is a path $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail \ldots$ where for some s_i along the path we have $M, s_i \models p$.
13. $M, s_0 \models \mathsf{A}(p\ \mathsf{U}\ q)$ iff for all paths $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail \ldots$ there is some s_i along the path, such that $M, s_i \models q$, and, for each $j < i$, we have $M, s_j \models p$.
14. $M, s_0 \models \mathsf{E}(p\ \mathsf{U}\ q)$ iff there is a path $s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail \ldots$, and there is some s_i along the path, such that $M, s_i \models q$, and, for each $j < i$, we have $M, s_j \models p$.

A temporal structure $M = (S, R, L)$ together with a state $s_0 \in S$ is a *(temporal) model* of a CTL formula p iff $M, s_0 \models p$. □

Example 6 (CTL Formulae)
Consider the temporal structure M of Example 4 (Figure 3). Some formulae that can be verified against M are:

1. $\mathsf{A}[\neg \mathit{test}\ \mathsf{U}\ \mathit{summary}]$

   On all paths (A) holds $\neg \mathit{test}$ until (U) *summary* holds, i.e. on all paths test unit must not occur before the summary unit occurs for the first time, and it is not relevant whether test unit does or does not occur afterwards.

2. $\mathsf{AG}((definition \wedge datastructure) \rightarrow \mathsf{AF}(example \wedge datastructure))$

   Whenever (AG) there is a definition of datastructure, there always must eventually occur (AF) an example of a datastructure.

□

Definition 7 (Temporal operator weak-until (W))
We extend the syntax of CTL (Definition 2) with a binary temporal operator weak-until (W). Its semantics is similar to the one of the until (U) operator with the difference that it is not required that the formula of the second operand holds at all in case that the formula of the first operand holds.
This extension results in two new CTL temporal connectives: AW (all paths weak-until), and EW (some path weak-until), which are defined as follows [6]:

$$\mathsf{E}[p\ \mathsf{W}\ q] := \mathsf{E}[p\ \mathsf{U}\ q] \vee \mathsf{EG}\ p$$

$$\mathsf{A}[p\ \mathsf{W}\ q] := \neg\mathsf{E}[\neg q\ \mathsf{U}\ (\neg q \wedge \neg p)]$$

□

Example 8 (AU vs. AW)
Recall the first formula of Example 6: $\mathsf{A}[\neg test\ \mathsf{U}\ summary]$. It requires that on every path summary unit occurs at some point in time. If we want to express that summary unit is optional and that the formula is also satisfied if $\neg test$ holds at each state along a path, then we use the CTL connective AW:

$$\mathsf{A}[\neg test\ \mathsf{W}\ summary]$$

□

4 Specification Patterns for Documents

The primary goal of the work described in this paper is the definition of a high-level specification formalism for consistency criteria for web documents. A pattern-based approach to the presentation, formulation, and reuse of property specifications in reactive systems has been introduced in [3]. A set of possible constraints has been defined and patterns have been created for them. The patterns are provided to the users who can identify similar requirements in their systems and select patterns that address those requirements. Until now, seven specification formalisms are supported, among

them CTL [1]. We found that many of these patterns could also be useful for expressing document properties [7, 8].
Because patterns defined in [3] are meant to be used by users familiar with the underlying specification formalism, user support for the specification process is not provided. Different from that situation, our use cases involve non-expert users; consequently, we have to support them in expressing formal consistency criteria. To this end, we provide an interface allowing to express loose criteria, which can later be enhanced if necessary.
As one can observe, criteria expressed in natural language are quite ambiguous. For example, requiring that *each definition of datastructure is followed by an example on the same topic*, does not specify precisely whether there should be an example of datastructure on all following paths after definition of datastructure, or whether it is enough having an example on some path. Natural language specifications of certain properties of specification patterns are also ambiguous. Here are some examples of such ambiguities:

- Does q *follows* p require that q has to hold on all following paths, or on some path?
- *After* s could mean after each s or after the first one. It is also not clear what happens if there is no s in the whole document. Is the criterion satisfied in this case or not?
- Does the meaning of *before* s include the narrative unit where s holds for the first time or not?

The ambiguities of natural language specifications were the main motivation for us to first define a set of *basic specification patterns* together with their corresponding CTL formulae and then to determine how the basic patterns can be modified, i.e. we defined a *set of modified patterns* with their corresponding CTL formulae. This way users can execute a two-stage process, first determining the general properties of the criterion they want to express adding refinements as necessary in the second step.
The semantics of pattern types, scopes, and modifiers we use is determined by the definition of the mapping of specification patterns onto CTL as will be detailed in section 5.

Example 9 (Properties of consistency criteria)
Let us consider the consistency criterion: *There always exists a summary unit before the first test unit.* The following important properties can be observed:

1. It expresses a kind of constraint: the *existence* of a summary unit.
2. It specifies the part of the document or, more precisely, of its temporal structure, where the specification should hold: *before* the first test unit.

The properties 1. and 2. characterize a specification pattern of the following kind: *Within the considered structure, on all paths starting from the current state property p holds before property s holds for the first time.* The considered structure can be the whole document, but also any document fragment. □

Definition 10 (Specification pattern)
A *specification pattern* (for documents) is a generalized representation of a commonly occurring requirement on the content and structure of documents (cf. [3]).
Specifications are instances of specification patterns.
A *specification pattern* is represented by a 4-tuple

$$(pattern\_type, p\_modifier, scope, s\_modifier).$$

- A *pattern type* (*pattern_type*) determines the type of the constraint expressed by the specification pattern. Each pattern type is represented by a *pattern type name* and one or two *pattern properties*. Pattern type names (universally, exists, follows, precedes) denote the type of the constraint and can only be understood in conjunction with pattern properties. A pattern property is a parameter which represents the CTL formula required to hold by the pattern type. Let p and q be CTL formulae. Possible values of *pattern type* are: universally p, exists p, q follows p, and p precedes q.

  universally p means that p holds in every narrative unit. exists p expresses that p has to hold in some narrative unit. q follows p means each unit satisfying p must be succeeded by a unit for which property q holds. p precedes q means that if property q holds in some narrative unit this unit must be preceded by a unit for which property p holds. By default, each pattern type applies to all paths of a document but this can be overridden.

- A *pattern modifier* (*p_modifier*) allows to refine a pattern type, by further restricting or loosening the original meaning. Possible values of *p_modifier* are: null_p, absence, $\mathsf{immediate}_p$, some_path. Modifier null_p indicates that the original meaning of a pattern type is not changed.

  Table 1 shows the allowed pattern modifiers for each pattern type. For pattern types universally p and exists p there is a pattern modifier some_path. It says that the constraint holds on some path of a document, as opposed to the default meaning. For the pattern type universally p a pattern modifier absence is defined, which denotes that p does not hold in any narrative unit. The pattern type q follows p can be used with the modifier $\mathsf{immediate}_p$, which expresses that q must hold in all next narrative units of the one where p holds.

pattern type	pattern modifiers
universally p	null_p , absence, some_path
exists p	null_p , some_path
q follows p	null_p , $\mathsf{immediate}_p$
p precedes q	null_p

Table 1: Pattern types with allowed pattern modifiers

- A *scope* determines where in a document a specification is intended to hold. A scope is represented by a *scope name* and one or two *scope properties*. A scope property is a parameter, which will be replaced by a CTL formula at instantiation time. Let s and r be CTL formulae. Possible values of *scope* are: globally, before s, after s, and between s and r.

  Scope globally requires no parameters and actually expresses an unrestricted scope - a specification having this scope applies to the whole document structure. before s expresses that the specification holds before or in the same narrative unit where s holds for the first time. Similarly, after s requires that the specification holds after or in the same narrative unit where s holds for the first time. Scope between s and r denotes each part of a document structure between an appearance of property s and the first following appearance of property r.

  Table 2 shows allowed combinations of pattern types and scopes. Every pattern type can be combined with scopes globally, before s, and after s. Pattern types universally p and exists p can also be used with scope between s and r.

pattern type	scopes
universally p	globally, before s, after s, between s and r
exists p	globally, before s, after s, between s and r
q follows p	globally, before s, after s
p precedes q	globally, before s, after s

Table 2: Pattern types with allowed scopes

- A *scope modifier* ($s\_modifier$) allows the refinement of a scope by further restricting or loosening the original meaning. Possible values of $s\_modifier$ are: null_s, real_before, and real_after. Modifier null_s indicates that the original meaning of a scope is not changed.

  Table 3 shows the allowed scope modifiers for each scope. Scope before

s can be restricted with a scope modifier real_before to express that the constraint expressed by the pattern type holds really before s, i.e. no later than in the preceding unit of the one at which s holds. Similarly, scope after s can be restricted with a scope modifier real_after to express that it is not sufficient that the constraint represented by the pattern type holds in the same unit with s, but only after it.

scope	scope modifiers
globally	null_s
before s	null_s , real_before
after s	null_s , real_after
between s and r	null_s

Table 3: Scopes with allowed scope modifiers

Specification patterns of the form (*pattern_type*, null_p, *scope*, null_s), where both modifiers are set to null, are called *basic specification patterns*, while the others are *modified specification patterns*.

□

According to Tables 1, 2, and 3 there are 45 specification patterns for documents, 14 of which are basic specification patterns (cf. Table 4 and 5 in section 5).

Example 11 (Basic specification patterns)
A temporal structure of a fragment of a WBT document about datastructures is depicted in Figure 4. The unit start is followed by a preliminary test about datastructures and an introductory example of a datastructure in the sequel. Thereafter, a definition and an example of datastructure follow. Further, users can proceed to optional units about abstract datatypes (definition and example of abstract datatypes). Afterwards a summary and a test about datastructure follow. Finally the end unit is presented. Users already familiar with the subject can, for the purpose of repetition, proceed to the summary of datastructure immediately after the start unit.
In total, there are three narrative paths through this structure:

p1 "Standard path" - for users who want to learn about datastructures without additional information:

$s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail s_3 \rightarrowtail s_4 \rightarrowtail s_7 \rightarrowtail s_8 \rightarrowtail s_9 \rightarrowtail ....$

p2 "Extended path" - for advanced users who are also interested in additional information about abstract datatypes:

$s_0 \rightarrowtail s_1 \rightarrowtail s_2 \rightarrowtail s_3 \rightarrowtail s_4 \rightarrowtail s_5 \rightarrowtail s_6 \rightarrowtail s_7 \rightarrowtail s_8 \rightarrowtail s_9 \rightarrowtail ....$

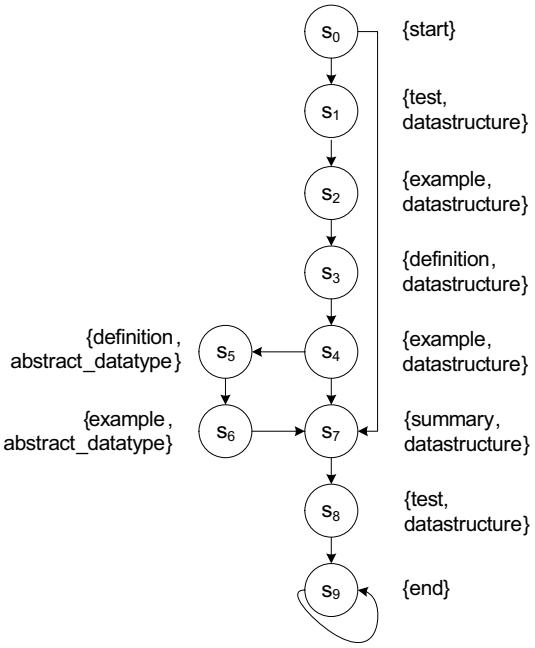

Figure 4: Temporal structure for Example 11

p3 "Repetition path" - for users already familiar with the content, for a brief repetition:

$s_0 \rightarrowtail s_7 \rightarrowtail s_8 \rightarrowtail s_9 \rightarrowtail ....$

Consider the following consistency criteria defined for the temporal structure shown in Figure 4:

c1 *There is always a test unit before the first definition unit.*

This criterion requires that a test exists, before the first definition. Obviously, the specification pattern of type exists p and scope before s is needed: (exists $test$, null_p, before $definition$, null_s). The corresponding CTL formula reads:

$$\mathsf{A}[\neg definition\ \mathsf{W}\ test]$$

c2 *Every definition of the topic datastructure is followed by an example of a datastructure.*

It is required that every definition of datastructure is followed by an example of datastructure. This corresponds to the pattern type q follows p and scope globally (this requirement concerns the whole document): $((example \wedge datastructure)$ follows $(definition \wedge datastructure)$, null_p, globally, null_s). The corresponding CTL formula reads:

$$\mathsf{AG}((definition \wedge datastructure) \rightarrow \mathsf{AF}(example \wedge datastructure))$$

c3 *Each unit between the start unit and summary of datastructure is dealing with datastructures.*

This criterion corresponds to the pattern type universally p and scope between s and r. Datastructure must hold within each narrative unit between the start unit and summary of datastructure: (universally *datastructure*, null_p, between *start* and (*summary* $\wedge$ *datastructure*), null_s). Note, that due to the pattern modifier null_p this pattern indeed requires the pattern formula to hold on all paths. The corresponding CTL formula reads:

$$\mathsf{AG}((start \wedge \neg(summary \wedge datastructure)) \rightarrow$$
$$\mathsf{A}[datastructure\ \mathsf{W}\ (summary \wedge datastructure)])$$

Criterion **c1** holds in the temporal structure in Figure 4. On paths **p1** and **p2** the first definition is found in unit s_3 and there is a test before it (unit s_1). On path **p3** there is no definition and thus the criterion holds by convention.
Also criterion **c2** holds in the temporal structure of Figure 4. There is one definition of datastructure (unit s_3) which is followed by an example on the same topic (unit s_4) on the relevant paths **p1** and **p2**. Note that there is also an example of datastructure before the definition, which does not affect the validity of criterion.
Criterion **c3** does not hold in the temporal structure in Figure 4. On path **p2** there are two narrative units (s_5 and s_6) between start unit and summary of datastructure at which datastructure does not hold. □

In the sequel we present some examples of modified specification patterns. To better explain the difference in the meaning between basic and modified specification patterns we also show the corresponding CTL formulae, which all can be found in Tables 4 and 5 (section 5).
The pattern modifier absence can be applied to the pattern type universally p to express that a certain property does not hold within any narrative unit.

Example 12 (Modifier absence)
To express the criterion *after the summary no definitions are allowed*, we use the pattern type universally p modified with pattern modifier absence (property definition should not hold) and scope after s. The resulting pattern

is (universally $definition$, absence, after $summary$, null_s). The corresponding CTL formula reads:

$$\mathsf{AG}(summary \rightarrow \mathsf{AG}\ \neg definition)$$

This criterion holds in the temporal structure of Example 11. □

In [3], absence is a separate pattern, but the CTL formulae are almost the same as in the case of the universally p pattern. The only difference is that the parameter p is negated. We believe it is important to first recognize the general type of constraint (something should hold in every narrative unit of some scope) and then to decide if it is a positive or negative property.
The pattern type modifier $\mathsf{immediate}_p$ can be applied to the pattern type q follows p. q follows p immediately means that for each unit, where p holds, q must also hold in all **next** narrative units.

Example 13 (Modifier $\mathsf{immediate}_p$)
Consider following constraints:

1. *Every definition of the topic datastructure has to be followed on all paths by an example on the same topic.* To express this constraint we use the pattern - $((example \wedge datastructure)$ follows $(definition \wedge datastructure)$, null_p, globally, $\mathsf{null}_s)$. The corresponding CTL formula reads:

   $$\mathsf{AG}((definition \wedge datastructure) \rightarrow \mathsf{AF}\ (example \wedge datastructure))$$

   The temporal operator F expresses that an example of datastructure holds eventually in some narrative unit.

2. *Every definition of the topic datastructure has to be immediately followed (i.e. in each next narrative unit) by examples on the same topic.* The pattern used above has to be modified with $\mathsf{immediate}_p$: $((example \wedge datastructure)$ follows $(definition \wedge datastructure)$, $\mathsf{immediate}_p$, globally, $\mathsf{null}_s)$. In the previous CTL formula the temporal operator F (eventually) is replaced by X (next):

   $$\mathsf{AG}((definition \wedge datastructure) \rightarrow \mathsf{AX}\ (example \wedge datastructure))$$

Both constraints hold in the temporal structure of Figure 4.

□

As already mentioned all pattern types apply, by default, to all paths of a document. If it suffices for a specification to hold on some path, we us the pattern modifier some_path. On the logic level, modifier some_path results in replacing a path quantifier A with E.

Example 14 (Modifier some_path)
Consider the criterion: *on all paths eventually a summary occurs*. This criterion is represented by the specification pattern: (exists *summary*, null_p, globally, null_s). The corresponding CTL formula reads:

$$\mathsf{AF}\ summary$$

The path quantifier (A) expresses that there is a summary unit on all paths. To express that *summary occurs eventually on some path*, we modify the pattern type exists p with some_path, i.e. we use the specification pattern: (exists *summary*, some_path, globally, null_s). In the previous CTL formula, the path quantifier "all paths" (A) is replaced by "some path" (E):

$$\mathsf{EF}\ summary$$

□

Example 15 (Modifier real_before)
Consider the criterion: *there is always a summary unit before the first test*. To represent it, we can use the specification pattern: (exists *summary*, null_p, before *test*, null_s). The corresponding CTL formula reads:

$$\mathsf{A}[\neg test\ \mathsf{W}\ summary]$$

The meaning of the scope before s implies that test and summary could actually hold in the same narrative unit. To express the more strict specification, that summary occurs really *before* test (no later than in the preceding unit) we use the specification pattern: (exists *summary*, null_p, before *test*, real_before). The corresponding CTL formula reads:

$$\mathsf{A}[\neg test\ \mathsf{W}\ (summary \wedge \neg test)]$$

□

5 Pattern Transformation to CTL Formulae

The meaning of a specification pattern is determined by its mapping onto a CTL formula. The mappings of specification patterns onto a CTL formulae are stored in the *table of mappings*. For every pattern, there is exactly one formula. Tables 4 and 5 show the entire table of mappings. The columns from 2 to 5 represent the specification pattern (pattern type, pattern modifier, scope, and scope modifier, respectively), and column 6 contains the corresponding CTL formula.

no.	pattern type	pattern modifier	scope	scope modifier	CTL formula
1	universally p	null_p	globally	null_s	$\mathsf{AG}\ p$
2	universally p	absence	globally	null_s	$\mathsf{AG}\ \neg p$
3	universally p	some_path	globally	null_s	$\mathsf{EG}\ p$
4	universally p	null_p	before s	null_s	$\mathsf{A}[(p \vee (\mathsf{AG}\ \neg s))\ \mathsf{W}\ s]$
5	universally p	absence	before s	null_s	$\mathsf{A}[(\neg p \vee (\mathsf{AG}\ \neg s))\ \mathsf{W}\ s]$
6	universally p	some_path	before s	null_s	$\mathsf{E}[(p \vee (\mathsf{AG}\ \neg s))\ \mathsf{W}\ s]$
7	universally p	null_p	before s	real_before	$\mathsf{A}[(p \vee (\mathsf{AG}\ \neg s))\ \mathsf{W}\ (s \wedge \neg p)]$
8	universally p	absence	before s	real_before	$\mathsf{A}[(\neg p \vee (\mathsf{AG}\ \neg s))\ \mathsf{W}\ (s \wedge p)]$
9	universally p	some_path	before s	real_before	$\mathsf{E}[(p \vee (\mathsf{AG}\ \neg s))\ \mathsf{W}\ (s \wedge \neg p)]$
10	universally p	null_p	after s	null_s	$\mathsf{AG}(s \rightarrow \mathsf{AG}\ p)$
11	universally p	absence	after s	null_s	$\mathsf{AG}(s \rightarrow \mathsf{AG}\ \neg p)$
12	universally p	some_path	after s	null_s	$\mathsf{AG}(s \rightarrow \mathsf{EG}\ p)$
13	universally p	null_p	after s	real_after	$\mathsf{AG}((s \wedge \neg p) \rightarrow \mathsf{AX}\ \mathsf{AG}\ p)$
14	universally p	absence	after s	real_after	$\mathsf{AG}((s \wedge p) \rightarrow \mathsf{AX}\ \mathsf{AG}\ \neg p)$
15	universally p	some_path	after s	real_after	$\mathsf{AG}((s \wedge \neg p) \rightarrow \mathsf{EX}\ \mathsf{EG}\ p)$
16	universally p	null_p	between s and r	null_s	$\mathsf{AG}((s \wedge \neg r) \rightarrow \mathsf{A}[p\ \mathsf{W}\ r])$
17	universally p	absence	between s and r	null_s	$\mathsf{AG}((s \wedge \neg r) \rightarrow \mathsf{A}[\neg p\ \mathsf{W}\ r])$
18	universally p	some_path	between s and r	null_s	$\mathsf{AG}((s \wedge \neg r) \rightarrow \mathsf{E}[p\ \mathsf{W}\ r])$
19	exists p	null_p	globally	null_s	$\mathsf{AF}\ p$
20	exists p	some_path	globally	null_s	$\mathsf{EF}\ p$
21	exists p	null_p	before s	null_s	$\mathsf{A}[\neg s\ \mathsf{W}\ p]$
22	exists p	some_path	before s	null_s	$\mathsf{E}[\neg s\ \mathsf{W}\ p]$
23	exists p	null_p	before s	real_before	$\mathsf{A}[\neg s\ \mathsf{W}\ (p \wedge \neg s)]$
24	exists p	some_path	before s	real_before	$\mathsf{E}[\neg s\ \mathsf{W}\ (p \wedge \neg s)]$
25	exists p	null_p	after s	null_s	$\mathsf{A}[\neg s\ \mathsf{W}\ (s \wedge \mathsf{AF}\ p)]$
26	exists p	some_path	after s	null_s	$\mathsf{A}[\neg s\ \mathsf{W}\ (s \wedge \mathsf{EF}\ p)]$

Table 4: Table of mappings, part 1

Every specification pattern is mapped onto exactly one CTL formula but not all CTL formulae can be represented in the form of a pattern instance. For example, there is no corresponding specification pattern for the following CTL formula: $\mathsf{AG}\ \mathsf{EF}$ *help* (*At any point help is eventually reachable*). This problem could be solved by introducing a new specification pattern, or by allowing the composition of existing patterns. However, there is a tradeoff between expressiveness and usability of the pattern system which we dealt with in favor of usability.

no.	pattern type	pattern modifier	scope	scope modifier	CTL formula
27	exists p	null_p	after s	real_after	$\mathsf{A}[\neg s\ \mathsf{W}\ ((s \wedge \neg p) \wedge \mathsf{AF}\ p)]$
28	exists p	some_path	after s	real_after	$\mathsf{A}[\neg s\ \mathsf{W}\ ((s \wedge \neg p) \wedge \mathsf{EF}\ p)]$
29	exists p	null_p	between s and r	null_s	$\mathsf{AG}((s \wedge \neg r) \rightarrow \mathsf{A}[\neg r\ \mathsf{W}\ p])$
30	exists p	some_path	between s and r	null_s	$\mathsf{AG}((s \wedge \neg r) \rightarrow \mathsf{E}[\neg r\ \mathsf{W}\ p])$
31	q follows p	null_p	globally	null_s	$\mathsf{AG}(p \rightarrow \mathsf{AF}q)$
32	q follows p	immediate_p	globally	null_s	$\mathsf{AG}(p \rightarrow \mathsf{AX}q)$
33	q follows p	null_p	before s	null_s	$\mathsf{A}[((p \rightarrow \mathsf{A}[\neg s\ \mathsf{U}\ q]) \vee \mathsf{AG}(\neg s))\ \mathsf{W}\ s]$
34	q follows p	immediate_p	before s	null_s	$\mathsf{A}[((p \rightarrow \mathsf{AX}q) \vee \mathsf{AG}(\neg s))\ \mathsf{W}\ s]$
35	q follows p	null_p	before s	real_before	$\mathsf{A}[((p \rightarrow \mathsf{A}[\neg s\ \mathsf{U}\ (q \wedge \neg s)]) \vee \mathsf{AG}(\neg s))\ \mathsf{W}\ s]$
36	q follows p	immediate_p	before s	real_before	$\mathsf{A}[((p \rightarrow \mathsf{AX}(q \wedge \neg s)) \vee \mathsf{AG}(\neg s))\ \mathsf{W}\ s]$
37	q follows p	null_p	after s	null_s	$\mathsf{A}[\neg s\ \mathsf{W}\ (s \wedge \mathsf{AG}(p \rightarrow \mathsf{AF}q))]$
38	q follows p	immediate_p	after s	null_s	$\mathsf{A}[\neg s\ \mathsf{W}\ (s \wedge \mathsf{AG}(p \rightarrow \mathsf{AX}q))]$
39	q follows p	null_p	after s	real_after	$\mathsf{A}[\neg s\ \mathsf{W}\ ((s \wedge \neg p) \wedge \mathsf{AG}(p \rightarrow \mathsf{AF}q))]$
40	q follows p	immediate_p	after s	real_after	$\mathsf{A}[\neg s\ \mathsf{W}\ ((s \wedge \neg p) \wedge \mathsf{AG}(p \rightarrow \mathsf{AX}q))]$
41	p precedes q	null_p	globally	null_s	$\mathsf{A}[\neg q\ \mathsf{W}\ (p \wedge \neg q)]$
42	p precedes q	null_p	before s	null_s	$\mathsf{A}[(\neg q \wedge \neg s)\ \mathsf{W}\ ((p \wedge \neg q) \wedge \mathsf{A}[\neg s\ \mathsf{W}\ q])]$
43	p precedes q	null_p	before s	real_before	$\mathsf{A}[(\neg q \wedge \neg s)\ \mathsf{W}\ ((p \wedge \neg q) \wedge \mathsf{A}[\neg s\ \mathsf{W}\ (q \wedge \neg s)])]$
44	p precedes q	null_p	after s	null_s	$\mathsf{A}[\neg s\ \mathsf{W}\ (s \wedge \mathsf{A}[\neg q\ \mathsf{W}\ (p \wedge \neg q)])]$
45	p precedes q	null_p	after s	real_after	$\mathsf{A}[\neg s\ \mathsf{W}\ ((s \wedge \neg p) \wedge \mathsf{A}[\neg q\ \mathsf{W}\ (p \wedge \neg q)])]$

Table 5: Table of mappings, part 2

6 Specification Tool

From the users' point of view it is not easy to express a consistency criterion and to choose the right pattern. Therefore, the proposed framework supports an incremental process for constructing the appropriate specification. Figure 5 shows a screen-shot of the GUI of our specification tool. Before

Figure 5: Specification tool

building the specification, the user chooses the document to be verified (no. 1 in Figure 5). After that, the process of constructing a specification starts. First, the user chooses the constraint type she wants to express (i.e. pattern type) - component 2 in Figure 5. For each pattern type, there is an explanation of its meaning. Second, a pattern modifier is to be set - component 3 in Figure 5. Only allowed modifiers for the previously chosen pattern type are enabled. The appropriate scope is to be chosen as the third component (no. 4 in Figure 5). The last component of the specification pattern is a scope modifier (no. 5 in Figure 5). Again, only the allowed scope modifiers for the already chosen combination of the pattern type and scope are enabled. After having chosen the complete specification pattern, the user is presented the natural language formulation of this pattern with placeholders (no. 6), which are to be bound to atomic propositions from the temporal model. For the inspection of the temporal model a dedicated additional tool is provided. Finally, the CTL formula corresponding to the constructed and refined specification is shown (no. 7). Having finished the specification, the user activates the model checker (no. 8).

Example 16 (Construction of a consistency criterion)
Assume the user wants to specify the following constraint: *On all paths there exists a summary unit before the first test unit. Summary unit and test unit may not occur in the same narrative unit.* The following steps are to be performed:

1. Choose the document to be verified (no. 1 in Figure 5).
2. Choose the pattern type exists p (no. 2 in Figure 5). This pattern type has one corresponding parameter (P) which will be instantiated in step 5.
3. Choose the pattern modifier null_p (no. 3 in Figure 5).
4. Choose the scope before s (no. 4 in Figure 5). This scope has one corresponding parameter (S) which will be instantiated in step 6.
5. Choose the scope modifier real_before (no. 5 in Figure 5).
6. The corresponding natural language phrase reads (no. 6 in Figure 5):

   On all paths, **P** holds eventually, before **S** holds for the first time.

   In our example, the user replaces **P** by the atomic proposition *summary* and **S** by the atomic proposition *test*.
7. Lookup the respective CTL formula from the translation table (cf. Table 4) and replace variables with atomic propositions determined in step 6:

$$\mathsf{A}[\neg test\ \mathsf{W}\ (summary \wedge \neg test)]$$

8. Verify if the specified criterion holds in the chosen document.

The steps 1 to 5 are performed by the user. In step 6 participate both the system and user, while the system performs steps 7 and 8.

□

7 Conclusion and Outlook

A user-friendly method for the high-level specification of consistency criteria for web documents has been presented. We use specification patterns for web documents to enable the users to incrementally build consistency criteria. This is especially convenient for users not familiar with temporal logics and can make all the difference between using temporal logic for consistency checking and ignoring it altogether.
In future work we will extend the expressive power of our patterns. For example, the following criterion cannot be expressed in CTL: *new concepts need to be defined before they are used*. To this end we will adapt our patterns to the temporal description logic ALCCTL [12]. We will also examine the possibility of composing specification patterns. First experiments let us expect that the proposed specification patterns and specification environment help users to formalize application-specific constraints on documents. However, to validate our method a field test with different user groups is planned.

8 Acknowledgments

This work has been funded by the German Research Foundation (DFG) under contract number FR 1021/7-1.

References

[1] Property Pattern Mappings for CTL. http://patterns.projects.cis.ksu.edu/documentation/patterns/ctl.shtml. Last visited Apr. 2008.

[2] M. Absmeier. Eine Validierungsumgebung für adaptierbare XML-basierte Dokumente. Master's thesis, Chair of Information Management, University of Passau, 2006. (in German).

[3] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. of the 21st int. conf. on software engineering*, pages 411–420. IEEE, 1999.

[4] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of theoretical Comp. Sci.: Formal Models and Semantics*, pages 996–1072. Elsevier, 1990.

[5] S. Flake, W. Mueller, and J. Ruf. Structured english for model checking specification. In K. Waldschmidt and C. Grimm, editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*. VDE Verlag, 2000.

[6] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.

[7] M. Jakšić. An approach to the example-based consistency checking of web documents. In *Proc. of the 18th Workshop on Foundation of Databases*, pages 75–79, Wittenberg, Germany, 2006.

[8] J. Köck. Musterbasierte Spezifikation von Dokumenteigenschaften. Master's thesis, Lehrstuhl für Informationsmanagement, Universität Passau, 2006. (in German).

[9] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *Proc. of the 27th ICSE*, pages 372 – 381, St. Louis, MO, USA, 2005. ACM Press.

[10] A. Pilger. Model-Checking von Temporalen Beschreibungslogiken durch Transformation in Prädikatenlogik erster Stufe. Master's thesis, Chair of Information Management, University of Passau, 2006. (in German).

[11] P. D. Stotts, R. Furuta, and C. R. Cabarrus. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *Information Systems*, 16(1):1–30, 1998.

[12] F. Weitl. *Document Verification with Temporal Description Logics*. PhD thesis, University of Passau, 2008. http://nbn-resolving.de/urn:nbn:de:bvb:739-opus-12528.

[13] F. Weitl and B. Freitag. Checking content consistency of integrated web documents. *Jour. of Comp. Sci. & Tech.*, 21(3):418–429, 2006.

[14] F. Weitl, M. Jakšić, and B. Freitag. Towards the automated verification of semi-structured documents. *Journal of Data & Knowledge Engineering*, 2008. accepted for publication.