Temporal Patterns for Document Verification

Mirjana Jakšić*and Burkhard Freitag

University of Passau
Mirjana.Jaksic, Burkhard.Freitag@uni-passau.de

Abstract

In this paper we present a novel user-friendly high-level approach to the specification of temporal properties of web documents which can be used for verification purposes. The method described is based on specification patterns supporting an incremental construction of commonly used consistency criteria. We show that our approach fills the gap between a temporal logic such as CTL as a powerful tool for specifying consistency criteria for web documents and users that maintain documents but have no or very limited knowledge about the specification formalism. An empiric assessment of the usability of specification patterns for web documents confirms that a pattern based specification shows significantly better results than the direct specification with CTL.

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 paper we address the problem of specifying consistency criteria for the purpose of verification of web documents. This work is part of the Verdikt project [19]. 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 document coherence and certain properties of the narrative flow, e.g., in e-learning or technical documentation.

In the Verdikt project the verification is performed by model-checking based on the temporal description logic ALCCTL [18]. For the sake of simplicity and clarity, we express consistency criteria in the more common, but also less expressive computation tree logic - CTL [6, 9] in this paper. However, the results described apply also to ALCCTL. 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, e.g. [17]. Applying a temporal logic such as CTL or ALCCTL 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 [5, 7, 14]. 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 [5] 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

*This work is partially funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant number FR 1021/7-2 proceed from low to higher complexity. The usability of specification patterns has been evaluated in comparison to a direct specification using plain CTL. The results show that for inexperienced users the specification with patterns is significantly easier than the one with CTL.

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, and
- evaluating the usability of the approach.

The paper is organized as follows. Section 2 describes the problem addressed. Section 3 introduces specification patterns for documents, section 4 deals with pattern transformation into CTL, while our specification tool is introduced in section 5. Evaluation results are presented in section 6. Section 7 discusses our results and related work. Section 8 concludes the paper.

2 Problem Description

Our aim is to check the consistency of the narrative structure of a document. The narrative structure represents relevant aspects of the content and structure in a document's model and is defined as follows (see also [19]).

Definition 1 (Narrative units. Start units. Narrative relation. Narrative path)**.** A document D is structured as a finite set $NU \neq \emptyset$ of *narrative units* where each $u \in NU$ is a cohesive, self-contained part of D. $SU \subset NU$ denotes a non-empty set of *start units* such that each $u \in SU$ is a sensible starting point for reading the document. A *narrative relation* NR on the set of narrative units is defined by $NR := \{(u, u') \in NU \times NU \mid$ it is sensible to proceed to unit u' immediately after having read unit $u\}$. Let $NR \subseteq NU \times NU$ be a narrative relation of a document. Then a (potentially infinite) sequence $(u_0, u_1, ...)$ of narrative units is a *narrative path* iff $(u_i, u_{i+1}) \in NR$ for each $i \in \mathbb{N}$.

Remark 2 (Narrative path)**.** Since web documents are typically not read linearly, narrative paths cannot be assumed to be acyclic. As a consequence, we have to consider narrative paths to be potentially infinite. By allowing infinite paths the document model does not put a limit on the number of times a certain narrative unit can be visited.

Definition 3 (Narrative structure)**.** A *narrative structure* is a tuple $NS = (NU, SU, NR)$ where NU is a set of narrative units, $SU \subseteq NU$ is a set of start units, and $NR \subseteq NU \times NU$ is narrative relation on NU such that the following holds:

i) NR is left-total on NU, i.e. for each $u \in NU$ there is some $u' \in NU$ such that $(u, u') \in NR$.
ii) Any narrative unit of NU can be reached on some narrative path in NR: for each $u \in NU$ there is a start unit $s \in SU$ and a narrative path $(u_0, u_1, ...) \in NR$ such that $u_0 = s$ and $u_i = u$ for some $i \in \mathbb{N}$.

Remark 4 (Narrative structure)**.** Demanding NR to be left-total simplifies the formal verification framework. For the sake of generality, narrative units which do not have any sensible successor unit are modelled as being reflexively related with themselves.

The process of reading a web document (by a human reader) can be modeled as a collection of paths in a state transition system (S, R, L) where the set of states S represents the narrative units of the document, $R \subseteq S \times S$ is a narrative relation on S and L represents a set of local interpretations, one for each state $s \in S$.

Definition 5 (Temporal structure. Temporal verification model)**.** A CTL temporal structure is a state transition system $M = (S, R, L)$, where:

- S is a set of states,
- AP denotes the set of all atomic CTL propositions,
- $R \subseteq S \times S$ is a left-total binary transition relation, defining the possible transitions between states,
- $L : S \mapsto \mathcal{P}(AP)$ is a labeling function, that assigns each state a set $I \subseteq AP$ of atomic CTL propositions that hold at this particular state.

A temporal verification model of a document is a temporal structure with a distinguished starting state $s_0 \in S$.

Example 6 (Narrative structure)**.** Figure 1a) depicts a fragment of a narrative structure of a web document taken from a web based training (WBT) about datastructures. The unit ”start” is followed by the ”definition of datastructures”. After this unit there are two possible branches to follow. The first one proceeds with an ”example of datastructures”, then with a ”summary” and a ”test about datastructures”, and finally with the ”end” unit. The other branch continues with a ”definition of abstract datatypes”, followed by an ”example of abstract datatypes”, and then joins the other branch at the ”summary” unit. Note, that the fact, that a human reader would probably classify the ”detour” over the ”abstract datatype” unit as a side note, is inessential in our context.

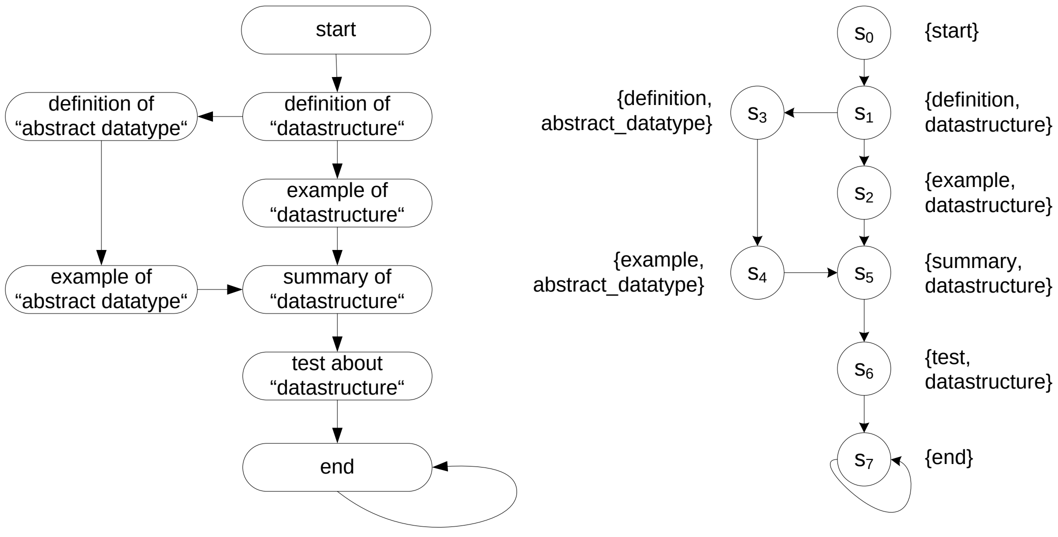

start
definition of “abstract datatype“
definition of “datastructure“
example of “datastructure“
example of “abstract datatype“
summary of “datastructure“
test about “datastructure“
end
a)
s0 {start}
{definition, abstract_datatype} s3
s1 {definition, datastructure}
s2 {example, datastructure}
{example, abstract_datatype} s4
s5 {summary, datastructure}
s6 {test, datastructure}
s7 {end}
b)

Figure 1: a) Narrative structure of a document, b) Temporal 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 1a). 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.

Example 7 (Temporal structure of a document)**.** Figure 1b) depicts the CTL temporal structure of a document with a narrative structure as shown in Figure 1a). In this structure holds:

- the set of states is defined as $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$,
- atomic propositions correspond to topics (datastructure, abstract datatype) and structural types (definition, example, summary, ...),
- the transition relation is given by $R = \{(s_0, s_1), (s_1, s_2), (s_1, s_3), ..., (s_6, s_7), (s_7, s_7)\}$,
- the labeling function determining the interpretation of a state is defined as $L = \{s_0 \mapsto \{start\}, s_1 \mapsto \{definition, datastructure\}, ..., s_7 \mapsto \{end\}\}$.

There are 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 $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrowtail s_4 \rightarrowtail s_5 \rightarrowtail s_6 \rightarrowtail s_7 \rightarrowtail ...$.

The main steps of consistency specification and verification are shown 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.

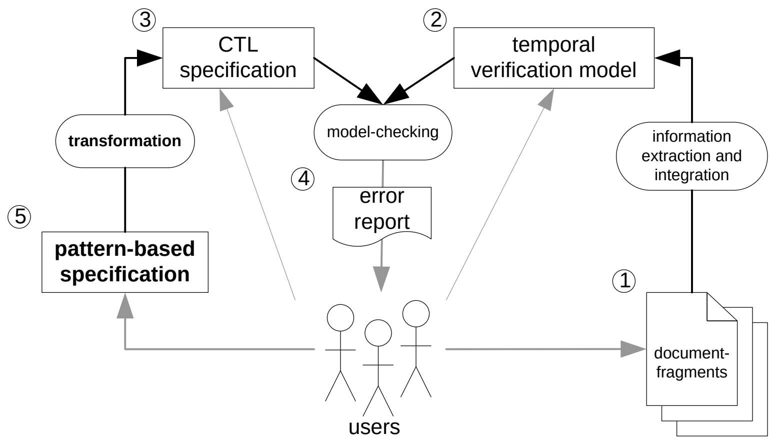

③
CTL specification
②
temporal verification model
transformation
model-checking
information extraction and integration
④
error report
⑤
pattern-based specification
①
document-fragments
users

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) 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). For example, a CTL formula which expresses the second criterion of Example 6 reads: $\mathsf{AG}((definition \wedge datastructure) \rightarrow \mathsf{AF}(example \wedge datastructure))$

Since CTL, as a temporal logic, 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 9). Specification patterns are translated into CTL formulae. Our approach to automated verification of semi-structured documents is presented in detail in [19].

3 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, which should fulfill the following properties:

- The proposed high-level formalism must represent the temporal properties of web documents and must be intuitively understandable, so that a user does not have to be aware of the underlying logic formalism.
- The system should provide a reasonable expressive power and yet stay compact and manageable.
- It is important to support the incremental development of specifications, i.e., it should allow the user to first recognize the general rule and then to refine it if required.
- The approach has to be extensible and adaptable to possibly different underlying logic formalisms.

A pattern-based approach to the presentation, formulation, and reuse of property specifications in reactive systems has been introduced in [5]. 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 [10, 13]. The abstraction from temporal properties allows users not to worry about the underlying logic. The flexible definition and organization of the original patterns allow us to choose only a subset and to adapt and extend it easily for our needs.

Because patterns defined in [5] 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 (see [15]) 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 be later enhanced if necessary.

Example 8 (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.

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 the "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 4.

Definition 9 (Specification pattern)**.** A *specification pattern* (for documents) is a generalized representation of a commonly occurring requirement on the content and structure of documents (cf. [5]).

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: $\texttt{null}_p$, `absence`, $\texttt{immediate}_p$, `some_path`. Modifier $\texttt{null}_p$ 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`

pattern type	pattern modifiers
`universally p`	$\texttt{null}_p$, `absence`, `some_path`
`exists p`	$\texttt{null}_p$, `some_path`
`q follows p`	$\texttt{null}_p$, $\texttt{immediate}_p$
`p precedes q`	$\texttt{null}_p$

Table 1: Pattern types with allowed pattern modifiers

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 $\texttt{immediate}_p$, which expresses that `q` must hold in all next narrative units of the one where `p` holds.

- 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 the original meaning. Possible values of *s_modifier* are: $\texttt{null}_s$, `real_before`, and `real_after`. Modifier $\texttt{null}_s$ 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.

Specification patterns of the form (*pattern_type*, $\texttt{null}_p$, *scope*, $\texttt{null}_s$), 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.

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

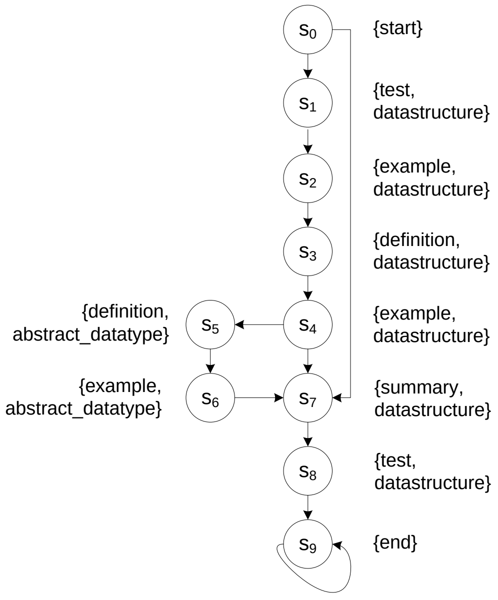


Figure 3: Temporal structure for Example 10

Example 10 (Basic specification patterns)**.** A temporal structure of a fragment of a WBT document about datastructures is depicted in Figure 3. 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:

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

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

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

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

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

Consider the following consistency criteria defined for the temporal structure shown in Figure 3:

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: ($\texttt{exists}\ test$, $\texttt{null}_p$, $\texttt{before}\ definition$, $\texttt{null}_s$). The corresponding CTL formula[1] 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): ($(example \wedge datastructure)\ \texttt{follows}\ (definition \wedge datastructure)$, $\texttt{null}_p$, $\texttt{globally}$, $\texttt{null}_s$). 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": ($\texttt{universally}\ datastructure$, $\texttt{null}_p$, $\texttt{between}\ start\ \texttt{and}\ (summary \wedge datastructure)$, $\texttt{null}_s$). Note that due to the pattern modifier $\texttt{null}_p$ this pattern indeed requires the pattern formula to hold on all paths. The corresponding CTL formula reads:

$$\begin{aligned}&\mathsf{AG}((start \wedge \neg(summary \wedge datastructure)) \rightarrow \\ &\qquad\qquad \mathsf{A}[datastructure\ \mathsf{W}\ (summary \wedge datastructure)])\end{aligned}$$

Criterion **c1** holds in the temporal structure in Figure 3. 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 3. 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 the criterion.

Criterion **c3** does not hold in the temporal structure in Figure 3. On path **p2** there are two narrative units (s_5 and s_6) between "start" 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.

Example 11 (Modifier immediate$_p$)**.** Consider the 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 - ($(example \wedge datastructure)$

[1] W - weak until; $\mathsf{A}[p\ \mathsf{W}\ q] := \neg\mathsf{E}[\neg q\ \mathsf{U}\ (\neg q \wedge \neg p)]$

follows $(definition \wedge datastructure)$, null$_p$, globally, null$_s$). The corresponding CTL formula reads: $\mathsf{AG}((definition \wedge datastructure) \rightarrow \mathsf{AF}\ (example \wedge datastructure))$

The temporal operator F expresses that an example of a 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 immediate$_p$: $((example \wedge datastructure)$ follows $(definition \wedge datastructure)$, immediate$_p$, globally, null$_s$). In the previous CTL formula the temporal operator F (eventually) is replaced by

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

Both constraints hold in the temporal structure of Figure 3.

Example 12 (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: $\mathsf{A}[\neg test\ \mathsf{W}\ (summary \wedge \neg test)]$

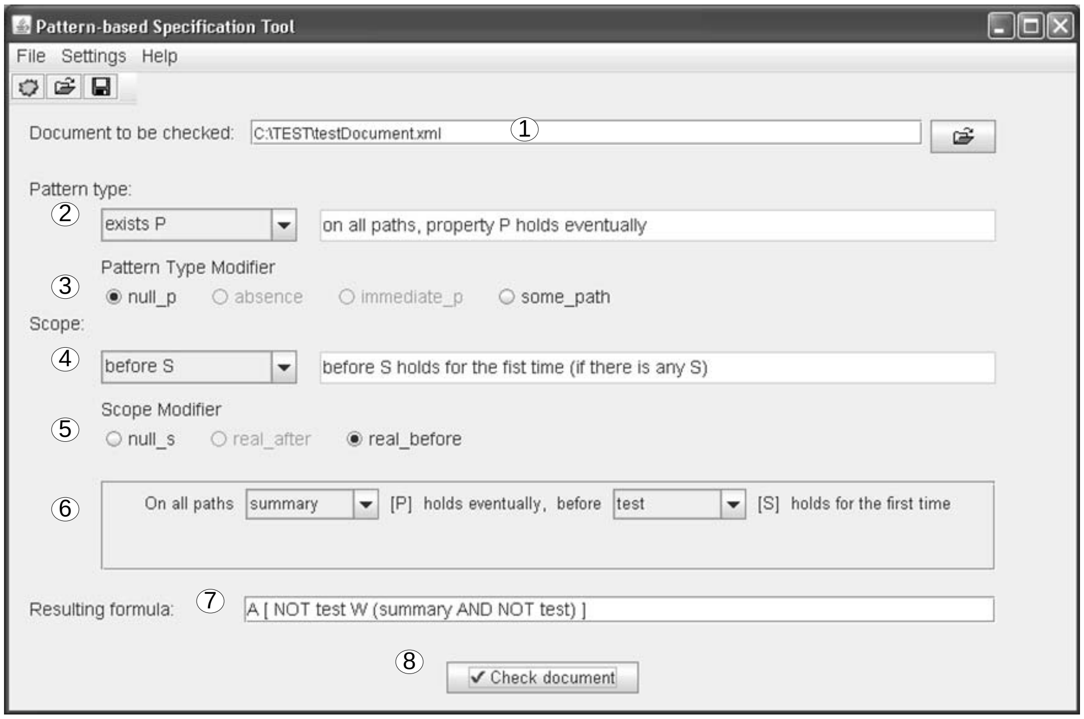
4 Pattern Transformation to CTL Formulae

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. Due to space constraints, Table 4 shows only a part of table of mappings with four patterns described in section 3. Columns one through four represent the specification pattern (pattern type, pattern modifier, scope, and scope modifier, respectively), and column five contains the corresponding CTL formula. The complete table of mappings can be found in [12].

pattern type	**pattern modifier**	**scope**	**scope modifier**	**CTL formula**
exists p	null$_p$	before s	null$_s$	$\mathsf{A}[\neg s\ \mathsf{W}\ p]$
exists p	null$_p$	before s	real_before	$\mathsf{A}[\neg s\ \mathsf{W}\ (p \wedge \neg s)]$
q follows p	null$_p$	globally	null$_s$	$\mathsf{AG}(p \rightarrow \mathsf{AF}q)$
q follows p	immediate$_p$	globally	null$_s$	$\mathsf{AG}(p \rightarrow \mathsf{AX}q)$

Table 4: Table of mappings (partial)

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.


Pattern-based Specification Tool

File Settings Help

1 Document to be checked: C:\TEST\testDocument.xml

Pattern type:

2 exists P — on all paths, property P holds eventually

3 Pattern Type Modifier: [✓ null_p] absence immediate_p some_path

Scope:

4 before S — before S holds for the fist time (if there is any S)

5 Scope Modifier: null_s real_after [✓ real_before]

6 On all paths summary [P] holds eventually, before test [S] holds for the first time

7 Resulting formula: A [NOT test W (summary AND NOT test)]

8 Check document

Figure 4: Specification tool

5 Specification Tool

A first prototype of a specification tool provides basic support for the specification process and helps users to incrementally build a specification pattern.

Figure 4 shows a screen-shot of the GUI. Before building the specification, the user chooses the document to be verified (no. 1 in Figure 4). 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 4. For each pattern type, there is an explanation of its meaning. Second, a pattern modifier is to be set - component 3 in Figure 4. 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 4). The last component of the specification pattern is a scope modifier (no. 5 in Figure 4). Again, only the allowed scope modifiers for the already chosen scope are enabled.

After having chosen the complete specification pattern, the user is presented with 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 [16]. 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 13 (Construction of a consistency criterion)**.** Let us assume that 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 4).

2. Choose the pattern type `exists p` (no. 2 in Figure 4). This pattern type has one corresponding parameter (P) which will be instantiated in step 5.
3. Choose the pattern modifier $\texttt{null}_p$ (no. 3 in Figure 4).
4. Choose the scope `before s` (no. 4 in Figure 4). 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 4).
6. The corresponding natural language phrase reads (no. 6 in Figure 4):

   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. Look up 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 both the system and user participate, while the system performs steps 7 and 8.

Our specification tool was implemented in Java 1.6. For model checking we used the CTL model checker NuSMV [4].

6 Evaluation

We evaluated the method of pattern-based specification of consistency criteria as compared to a direct specification using plain temporal logic CTL. Goals of the evaluation were:

- to show that even inexperienced users, after receiving instructions about specification patterns and their usage, can successfully use them,
- to show that under comparable conditions, the application of specification patterns by inexperienced users leads to remarkably better results than the usage of plain temporal logic like CTL.

The evaluation has been conducted with 108 volunteer participants, all of which were students from various fields of study. The participants were split into two groups of 54 members each. No participants had previous experience with either CTL or specification patterns for web documents.

Test questions addressed the formulation of consistency criteria concerning a single test document. Both groups had to specify the same five consistency criteria. The first group (control group) was asked to specify criteria using CTL whereas the second group (experimental group) had to apply specification patterns. The test document was a user manual for a digital camera, found on the manufacturer web site; for details see [11].

The test was performed separately for each group. At the beginning, both groups were given an introduction to the Verdikt project and the test environment. Afterwards, the first group attended a practical compact training course in CTL (ca. 45 minutes). The CTL-syntax and semantics were explained on a rather intuitive level relying mostly on examples and graphical illustrations. Some examples shown had a structure similar to the test questions. After the CTL training, the participants of the first group were asked to answer the test questions separately and individually, i.e., without team work. An overview of CTL operators was available to each student for reference.

The second group was introduced to the structure and meaning of specification patterns as well as examples of their usage (ca. 30 minutes). As for the first group, some examples shown had a structure similar to the test questions. As reference material a list of all specification patterns was available.

We validated the answers as either usable or unusable. A specification was considered usable, if no ”false positives” resulted when using it in a verification run. However, we classified as acceptable ”false negatives” that were produced when applying a specification that was usable in the aforementioned sense. That is, specifications that were stricter than required were classified as usable and weaker specifications were classified as unusable.

The validation results confirm the usability of our approach. The results show that after having received brief instructions about specification patterns even inexperienced users can use them with a success rate of over 70%. The validation results confirm also that for inexperienced users it is considerably easier to express the criteria by using specification patterns as compared to using CTL formulae. Under comparable conditions, participants of the first group (CTL) could only answer ca. 32% of all questions correctly, while for the second group the success rate was over 70%. Precise validation results are presented in Table 5 and in the diagram of Figure 5.

# correctly answered	**# participants from**	
questions per participant	**group 1 (CTL)**	**group 2 (patterns)**
5	0	18
4	4	15
3	8	10
2	22	5
1	18	0
0	2	6
totally participants	**54**	**54**

Table 5: Validation results

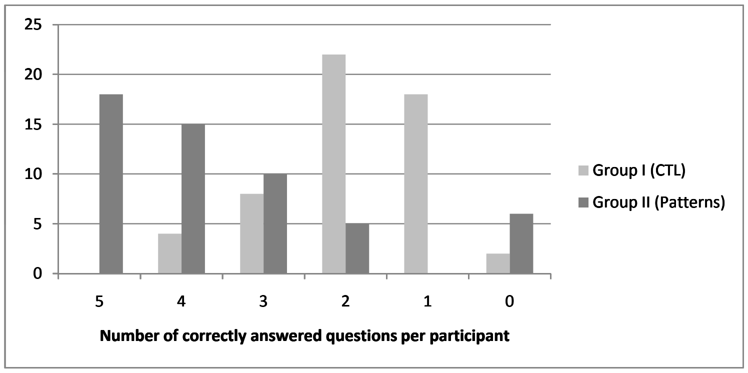


Figure 5: Number of correctly specified criteria per participant

The validation of test results has also confirmed our assumption that the instantiation of parameters should be additionally supported. This problem is addressed by the Verdikt project [16].

7 Discussion

The described temporal patterns with CTL as the underlying formalism are adequate for expressing path-oriented temporal criteria for web documents. Criteria also containing semantical dependencies, like, e.g., *every definition of some topic is on all succeeding paths followed by an example of the same topic*, cannot be expressed in CTL. Within the Verdikt project, we use a temporal description logic ALCCTL [18] as a specification formalism. Our patterns can also be used with ALCCTL [19], but in this case the instantiation of parameters becomes rather complex and therefore has to be additionally supported. To this end, in our ongoing research we are extending the method described here towards a user-friendly representation of ontological knowledge.

If a criterion does not hold in the temporal structure, the model checking results in a rather technical counter-example, which is not very informative for a user. For this reason, an incremental construction of counterexamples has been defined [20].

The prototype of the specification tool presented in section 5 shows only one rather simplified way to support the usage of specification patterns. Our ongoing research considers also an example-based specification method.

Our approach to document verification by model checking is not targeted at the XML data model of ordered trees but at documents with a graph (but not necessarily tree) structure such as hypertext. Properties of paths in such graph-structured documents are hard to express and inefficient to check using XPath and first order logic. A detailed study of the expressiveness and performance of our approach as compared to methods based on XML processing is presented in [18].

The problem of high-level specification appears in different research areas and there are also diverse approaches to its solution. Diagram-based languages have been suggested in the areas of real time systems [2] and workflow modeling [3]. These graphical languages are closely related to the underlying formalism and, as a result, it is impossible to interpret or create diagrams without deep knowledge about the logic they represent.

Dwyer et al. were the first to present specification patterns for temporal properties [5]. Many other researchers dealing with temporal specifications have adapted these patterns to different purposes. In the sequel we refer to two of them. In [7] the authors adopted the idea of specification patterns. An interactive visual framework that employs structured English sentences as a user front-end for the specification of Clocked CTL (CCTL) formulae for model-checking has been developed. [14] deals with timing-based requirements for embedded systems. The authors present real-time specification patterns in terms of three commonly used real-time temporal logics (MTL, TCTL, RTGIL). In addition, they have developed a structured English grammar, to further facilitate the understanding of a specification meaning. In the application field of web documents, which could possibly have a very complex structure with many branches, the representation of patterns only by structured language is not sufficient. Our ongoing research considers an additional example-based support for the specification process.

Conceptual authoring [8] is a method which uses a natural language text as a means of presenting semantic content during knowledge editing. By this method all editing operations are defined directly on an underlying logical representation, governed by a predefined ontology. By using common generic phrases users need not be aware of the underlying formalism. We will evaluate conceptual authoring within our ongoing research concerning parameter instantiation, as mentioned above.

8 Conclusion and Outlook

A user-friendly method for the high-level specification of consistency criteria for web documents has been presented and its usability has been shown. We define specification patterns for web documents as a high-level formalism for consistency criteria. Patterns hide the underlying logic formalism and are represented by simple natural language expressions, like, e.g., *existence before*. They also allow for the incremental building of specifications. 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. The usability of our approach has also been demonstrated. We believe that we found a good balance between expressive power and usability. If necessary, the system can be extended with new patterns and also adapted to another underlying formalism.

In future work we will adapt our patterns to the temporal description logic ALCCTL [18] to increase their expressive power. First experiments let us expect that the proposed specification patterns and specification environment help users to formalize application-specific constraints on documents. We will also examine the possibility of composing of specification patterns. The usability of patterns is going to be increased by an example-based specification method.

9 Acknowledgments

Dr. Petia Genkova, University of Passau, contributed to the evaluation with valuable insights.

References

[1] Property Pattern Mappings for CTL. http://patterns.projects.cis.ksu.edu/documentation/patterns/ctl.shtml. Last visited March 2010.

[2] A. Del Bimbo, L. Rella, and E. Vicario. Visual specification of branching time temporal logic. In *Proc. of the 11th Int. IEEE Symp. on Visual Languages*, pages 61–68. IEEE, 1995.

[3] Marco Brambilla, Alin Deutsch, Liying Sui, and Victor Vianu. The role of visual tools in a web application design and verification framework: A visual notation for LTL formulae. In *Proc. of ICWE 2005*, volume 3579 of *LNCS*, pages 557–568. Springer, 2005.

[4] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[5] Matthew B. Dwyer, George S. Avrunin, and James 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.

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

[7] 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.

[8] Catalina Hallett, Donia Scott, and Richard Power. Composing questions through conceptual authoring. *Computational Linguistics*, 33(1):105–133, 2007.

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

[10] Mirjana 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.

[11] Mirjana Jakšić. Evaluation eines Ansatzes zur Muster-basierten Spezifikation von Konsistenzkriterien fuer Web-Dokumente. Technical Report MIP-0906, University of Passau, Germany, 2009. http://www.fim.uni-passau.de/fileadmin/files/forschung/mip-berichte/mip-0906.pdf.

[12] Mirjana Jakšić and Burkhard Freitag. Temporal Patterns for Document Verification. Technical Report MIP-0805, University of Passau, Germany, 2008. http://www.fim.uni-passau.de/wissenschaftler/forschungsberichte/mip-0805.html.

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

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

[15] Christian Schönberg, Mirjana Jakšić, Franz Weitl, and Burkhard Freitag. Verification of web content: A case study on technical documentation. In *Proceedings of WWV 09*, pages 53–68, Linz, Austria, 2009.

[16] Bernhard Stadler. Dokumentmodellierung mit Ontologien. Bachelorarbeit, Lehrstuhl für Informationsmanagement, Universität Passau, 2008.

[17] P. David Stotts, Richard Furuta, and Cyrano Ruiz Cabarrus. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *Information Systems*, 16(1):1–30, 1998.

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

[19] Franz Weitl, Mirjana Jakšić, and Burkhard Freitag. Towards the Automated Verification of Semi-structured Documents. *Data & Knowledge Engineering*, 68:292–317, 2009.

[20] Franz Weitl and Shin Nakajima. Incremental construction of counterexamples in model checking web documents. In *Proceedings of WWV 2010*, Wien, Austria, 2010.