Hyperdocuments as Automata: Verification of Trace-based Browsing Properties by Model Checking*

P. David Stotts
Computer Science Department
University of North Carolina
Chapel Hill, NC 27599-3175
stotts@cs.unc.edu

Richard Furuta
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
furuta@cs.tamu.edu

Cyrano Ruiz Cabarrus
School of Systems Information and Computer Science
University Francisco Marroquin
Guatemala

Abstract

We present a view of hyperdocuments in which each document encodes its own browsing semantics in its links. This requires a mental shift in how a hyperdocument is thought of abstractly. Instead of treating the links of a document as defining a static directed graph, they are thought of as defining an abstract program, termed the *links-automaton* of the document. A branching temporal logic notation, termed HTL*, is introduced for specifying properties a document should exhibit during browsing. An automated program verification technique called *model checking* is used to verify that browsing specifications in a subset of HTL* are met by the behavior defined in the links-automaton. We illustrate the generality of these technique by applying them first to several Trellis documents, and then to a Hyperties document.

CR Categories: I.7.2 (Hypertext/hypermedia), H.5.1 (Hypertext navigation and maps), D.2.2 (Petri nets), D.2.4 (Program verification), F.3.1 (Specifying and Verifying and Reasoning about Programs).

Key words: Verification, hypertext, hypermedia, browsing semantics, temporal logic, model checking, Petri nets.

*This work is based upon work supported by the National Science Foundation under grant numbers IRI–9007746 and IRI–9015439.

1 The problem and the approach

Among others, Halasz has noted the need for structural search and query mechanisms for increasing the utility of hypertext documents. He identifies two major subtasks [20]: "...to design a query language geared toward describing hypermedia network structures," and "...implementation of a search engine capable of satisfying the queries expressible in the new language." This report describes an approach to both of these subtasks. We concentrate on a mechanism for answering queries concerning a document's dynamic properties, that is, what sequences of links a reader may be allowed to follow during browsing. However, the basic technique, that of *model checking* (borrowed from concurrent system verification), can be easily adapted to locating readers within the structure under examination.

The work described deals with hypertexts that are best thought of as cohesive, consistently structured, non-linear documents rather than as accumulated collections of information. We see a current and future need for hypertextual documents that "tell a story": training scripts, tutorials, interactive writing/fiction, descriptive/persuasive texts...these are a few examples of the class of hypertext under discussion in this paper. Many uses also exist for hypertexts that are not "writing" *per se*, but nonetheless are structured documents that have an identifiable author (one person or a tightly-coupled collaboration) and in which some constraints, prescriptions, and proscriptions need to apply to the linkages among elements.

In this view, a hypertext is an *interactive document*, providing a non-linear, dynamic analogue to the traditional notion of structured document. In an interactive setting, the notion of a document's structure must extend beyond the normal static concept of its graph (trees, typically, for paper-based documents) of components into the dynamic domain of browsing. A formal, analyzable description must be available of how those components might be presented to a reader—the possible sequences and parallel threads of activity within the document. We refer to this as the *dynamic structure* of an interactive document [16], as contrasted with the notion of static structure provided by a collection of links.

This report presents an approach to expressing dynamic properties an author may want in a document, and provides a method for verifying in an automated fashion whether the linked structure of a document satisfies the required property specifications. The emphasis here is on behavior that is allowed by *links alone*, independent of any navigation aids that a browser or navigation program might provide. Such knowledge allows an author to build a structure to offer the desired linkages no matter what system is used to accessed the document. Portions of this work previously have been reported in preliminary form [42].

Links-only document behavior

Trellis is a general man/machine interaction model that has been used previously as the basis for various hypertext systems and experiments [40, 41, 39]. Research on Trellis has concentrated on identifying the advantages of using formal automata to define hyperdocuments. Though we first explain our verification method in the context of Trellis, we would like to emphasize up front that the approach is applicable to documents produced by any hypertext system. What is required is a particular way of thinking about a document — that is, one must view a document as an abstract automaton that specifies the process of browsing within it. Such a view is easily obtained for the hypertext systems in use today. In fact, for systems other than Trellis, the linked structure of a document can usually be thought of as the state transition diagram of a finite state machine (FSM). For Trellis, the links define a more powerful class of automaton called a *place/transition net* (PT-net, also called Petri net). No matter what its power, we will

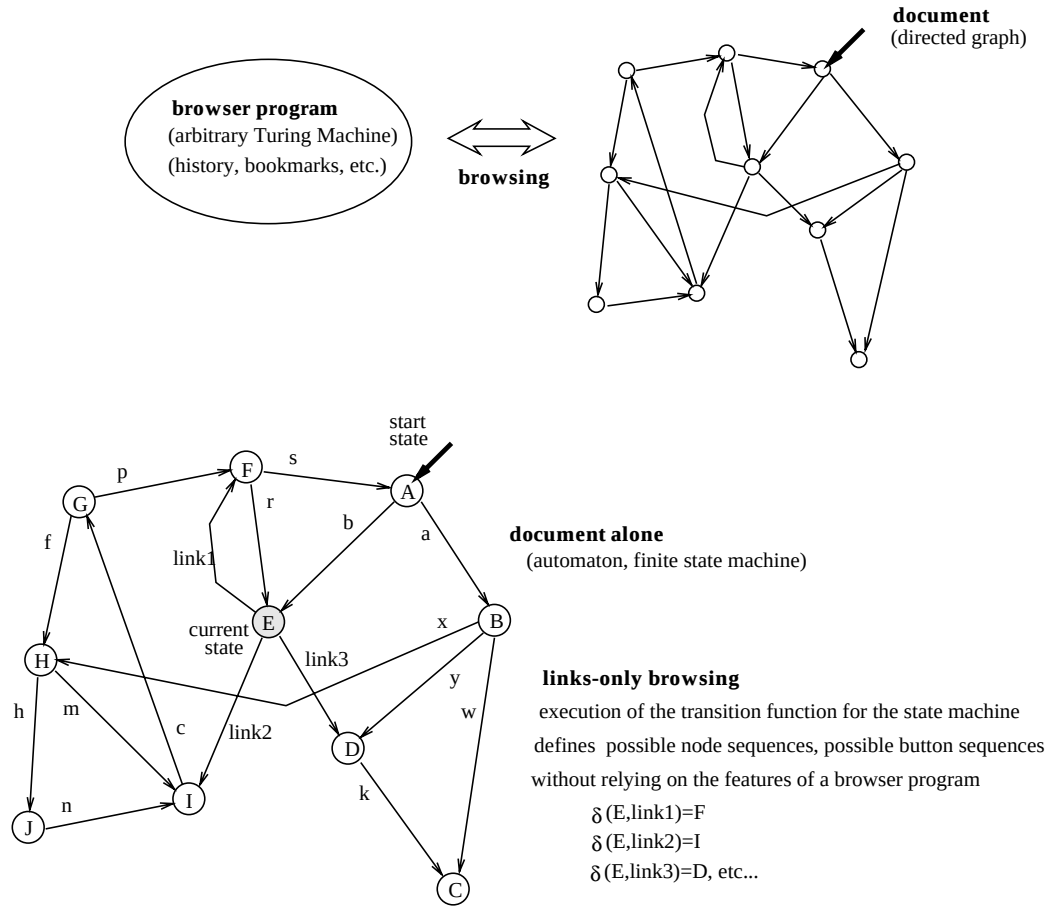

Figure 1: Traditional view of hypertext document, and automaton view.

refer to the automaton inherently defined by a document's hyperlinks as its *links-automaton*.

This conceptual framework is illustrated in Figure 1. The top half shows the traditional view of a hyperdocument — a browser program allowing navigation of a directed graph. The bottom half shows the links-automaton view, in which the directed graph is treated as an FSM. In this links-automaton, it is clear that there exists a browsing path from the starting node (A), continuing through the nodes E and D, and ending with node C. The links-only behavior of the document does not allow any further browsing from this point, because there is no transition out of node C. In order for browsing to continue, the author of this structure must be relying on some feature of the browser (such as backup, general history, restart, or bookmark at the table-of-contents, for example) to "warp" the reader to another location within the document. In order to guarantee behavior that does not require browser features, an author may wish to ensure that there is some path from each node back to the start node, for instance.

Other research efforts to formalize some aspects of browsing, such as the HAM [7], have focused on removing the arbitrary nature of the Turing machine that provides browsing services for a directed graph. In Trellis, we have extended the fundamental power of the links-automaton itself. The browser for a Trellis system is a simple program — limited to a strict implementation

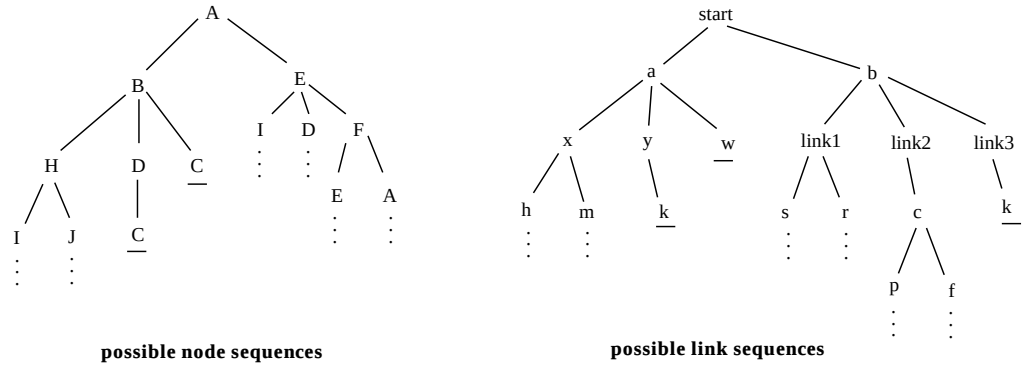

Figure 2: Branching event trees for links-automaton in Figure1

of the transition rule for PT-nets. Trellis hyperdocuments have more power and expressibility than FSM-based documents when considering the behavior allowed in their links alone. All behavior that can be expressed in FSMs can be expressed in PT-nets; in addition, a PT-net inherently allows parallel threads of activity (obtained with multi-head/multi-tail links), and is not limited to expressing a finite number of states.[1] In our most recent Trellis model, an even more powerful net-based formalism allows a hyperdocument to encode the interactions of multiple cooperating readers, as needed for collaboration-support systems [17]. Section 3 discusses such documents briefly.

Formalizing browsing properties

Suppose we are given a hypertext containing among other things content elements X and Y and buttons B and C (link anchors). We would like to formalize statements like "all (sufficiently long) browsing sessions must encounter X, but only sometime after seeing Y," or "there is at least one browsing session encountering Y," or "there is a browsing session in which at some point buttons B and C are both selectable." More general specifications include "there must be some path from any node X back to the index," and "every node Y must have some out links."

The links-automaton of a hyperdocument is an abstract program, and it can be thought of as generating a tree of possible event sequences (either sequences of button clicks, or sequences of content displays, depending on the properties to be studied). Figure 2 illustrates event sequences for the links-automaton in Figure 1; in each tree shown, an underlined node has no children. The tree on the left shows the possible sequences of content displays whereas the tree on the right shows possible sequences of button selections. Consider the left tree in Figure 2. One possible (finite) sequence of content displays is nodes A, B, C; another possible sequence is A, B, H, J, ... and on, perhaps not terminating.

We will formalize the browsing behavior allowed in a hyperdocument as the collection of possible event traces produced by the links-automaton of the document. We will specify properties we want the traces to exhibit with a *branching temporal logic*.[2] Finally, we will analyze the document for the presence or absence of properties with a program verification technique called *model checking*.

[1] We assume some familiarity with general net theory. Interested readers can get details in previous Trellis papers [40], and in summary texts by Murata [32], Reisig [36], or Peterson [33].

[2] The concept of time implied by the term "temporal" is not duration but rather the relative ordering of events in a sequence.

Background in temporal logic

Work in temporal logic was first conducted under the name of tense logic by symbolic logicians and philosophers for reasoning about ordering of events in time without mentioning time explicitly. In the last decade, temporal logic has become a convenient formalism used, among other things, in program verification [6, 35], in artificial intelligence and cognitive science [22, 21, 25], in specification and verification of concurrent computations [2, 29], in verification of network protocols and sequential circuits [31], and in verification of software requirements specifications [1]. A complete survey of classical works in temporal logic as well as its current applications is found in [19]. More recently, temporal logic has been applied successfully to the specification, verification, and analysis of reactive systems, which are concurrent computations that maintain a relationship with their environments; an overview of this area can be found in Manna [28]. Hyperdocuments, especially when interpreted as concurrent computation (as in Trellis) bear some resemblance to reactive systems.

In a branching temporal logic one introduces special logic symbols that allow formulation of assertions involving relative ordering as well as quantification over paths in a tree-like model of time. Various authors have proposed distinct syntaxes and semantics for branching time logics [24, 4], which differ in expressive power. However, CTL*, developed by Emerson and Halpern [13], is one of the most general languages for branching time logics, since it properly contains most of the other branching time logic languages. We have chosen CTL*as as the basis for our hyperdocument notation, called HTL*, or *hypertext temporal logic*.

The work of Gabbay, *et al.* [18] shows that CTL*, (and so HTL*) is expressively complete in the sense that all first-order logic properties can be represented by temporal formulae. However, not all browsing properties of hypertext are first-order logic properties; this is one of the weaknesses of our current approach. Similar limitations in other contexts have inspired *extended temporal logics*, such as Wolper's addition of new operators associated with regular expressions [43]. We expect our work could be similarly extended, but we will not discuss the topic further here.

Application of temporal logic to nets and hyperdocuments

One of the novel contributions of this work is the treatment of the coverability graph of a PT-net as a finite state machine, making model checking of the net states possible. Sinachopoulos also has considered the application of temporal logic to the study of place/transition nets [38], exploring the differences between branching time and partial order logics. It is suggested to apply UB [4], CTL [15], CTL* [13] and POTL [34] to the study of the firing sequences in place/transition nets. Emphasis is on an axiomatic specification approach; he does not consider model checking as a method. Beeri *et al.* [3] have presented a method for using a modal (temporal) logic for structural queries in a hyperdocument. The work is unrelated to nets, but looks at hyperdocuments as directed graphs. The goal of a query is to locate a particular subgraph. Queries in the temporal logic are essentially structural patterns to be matched against the document graph.

Contributions of this work

This report presents a mixture of new results from the authors' research, prior work of the authors, and prior work of other researchers. We would like to clearly distinguish these components before getting into the technical presentation.

Our novel contributions are:

- the idea of a links-automaton, in which we view a hyperdocument as an abstract process instead of a static data structure;
- the use of temporal logic for expressing browsing properties of hyperdocuments, and the use of the links-automaton of a hyperdocument as the basis for model checking of these properties;
- the HTL* temporal logic notation for succinctly expressing hyperdocument browsing properties, and the HTL subset of HTL* that can be efficiently verified with model checking;
- the idea of using the reachability graph of a PT-net as the finite state model for checking properties of Trellis hyperdocuments.

Prior work on which we build includes:

- the Trellis hypermedia system (developed by the authors), which we use for example documents to analyze;
- the Hyperties system from Shneiderman [37, 30], which we also use for sample documents to analyze;
- CTL*, CTL, and POTL temporal logic notations, which we employ as the semantic basis for HTL* and HTL;
- the model checking algorithms and software from Clarke, which we used to implement our hyperdocument analysis tools.

We have not developed any new model checking techniques, but we have shown how to apply this useful work from programming languages research to the domain of hypermedia. We have also shown how to derive the necessary models from hyperdocuments, using two very different systems as examples (Trellis and Hyperties). We have not extended the basic power of temporal logic languages over that of CTL* and POTL, but we have defined specific temporal operators for more naturally expressing the browsing properties of hyperdocuments. Finally, our techniques are general, applying to any hypermedia system including the suddenly ubiquitous World Wide Web.

In the remainder of this paper we first present the HTL temporal logic notation (the full HTL* notation is presented in the appendix); we show how to encode hyperdocument browsing properties in HTL; we explain how to use Clarke's model checking techniques to verify these properties for hypermedia, including how to extract a links-automaton from a hyperdocument; we illustrate our methods on documents from Trellis and Hyperties; and we discuss work remaining to be done to build on our results.

2 Hypertext Temporal Logic

This section discusses two related temporal logic notations: HTL* and HTL. HTL* is a fully expressive language with operators designed to succinctly express browsing concepts and properties common in hyperdocuments. HTL is a subset of the HTL* notation; the properties expressed

in HTL can be very efficiently checked for validity using a *model checking* algorithm developed by Clarke [11]. Our current model checking experiments are restricted to the HTL subset, so we have put the full definition of HTL* in Appendix A. Readers familiar with basic temporal logic should be able to skip the appendix and still understand the main results of this work.

The semantics of HTL* is closely related to CTL* [13]. The syntax has been altered to more directly reflect the hypertext domain, using a different notation and including backward path quantification to more conveniently express the kinds of statement authors need to make about how documents are to be browsed. As discussed in section 1, we think of a hyperdocument as defining many possible sequences of node visits (or link followings), represented as a tree starting at the first node browsed. At each node is a browsing sequence, a reader has many choices as to what the next node visited will be, producing the branch points in the tree. We also assume that there are many different basic properties of node that are of interest during analysis (such as "the page is a cgi-script" or "the page contains an image map" or "the page has no out-links"). At each node, these basic properties will be either true or false. We speak of this collection of basic properties as the *state language*, a set of atomic predicates.

As with CTL*, HTL* formulae add path quantifiers and sequence operators to a basic predicate calculus over the state language, allowing formulae expressing properties that hold over the *tree* of browsing possibilities itself:

- $\vec{\exists}$ is a path quantifier saying "there exists a forward path"
- $\vec{\forall}$ is a path quantifier saying "for all forward paths"
- $\Diamond$ is a sequence operator saying "eventually (in some state from here)"
- $\Box$ is a sequence operator saying "always (in all states from here)"
- $\circ$ is a sequence operator saying "in the next state"

In our notation the temporal operators have higher precedence than the traditional logical connectives. We use parentheses where extra clarity is needed. Among operators at the same level of precedence, we apply them left to right in a formula.

The formula "$\vec{\forall} \Diamond$ P" says that in all browsing paths through a document (going in a forward direction) the reader will eventually encounter a state (node, page) in which the predicate P is true (where P is some predicate from the state language). Several other operators are explained in Appendix A (*e.g*, backward path quantifiers, past-tense and future-tense operators, etc.); we leave the reader to study those for a formal treatment, and present the basics of HTL* informally via the following examples interpreted in the context of hyperdocument browsing.

Examples of browsing properties in HTL*

We now present a list of formulae in the context of browsing a hyperdocument. The basic state language of HTL* in our experiments is simple, containing only assertions about which content elements are viewable (*e.g.*, c.introduction) and which links (buttons) are selectable (*e.g.*, b.next). We assume that each formula refers to the initial state of the document, *i.e.*, these are properties that hold when browsing *begins*.

- $\vec{\forall} c$.hello means "At the beginning of every browsing session, you must encounter the *hello* screen."

- $\vec{\forall}\Box\vec{\exists}\circ c$.end means "Wherever you are, you can immediately leave the document" (assuming this is the interpretation of content *end*).

- $\vec{\forall}(\Diamond\varphi \vee \Box\neg\varphi)$ is a tautology which states that in all browsing sessions either φ happens at least once or it never happens at all.

- $\vec{\forall}\Box\vec{\exists}\Diamond c$.menu means "wherever you happen to be, it is possible to eventually get back to the *menu*."

- $\vec{\forall}\Box$(c.error $\leftrightarrow$ c.description) means "for all browsing sessions, it is always the case that content element *error* is visible if and only if *description* is visible."

- $\vec{\exists}\Diamond\vec{\forall}\Box(\neg b$.info) means "it is possible for button *info* to eventually die, that is, to never again be selectable."

- $\vec{\forall}\Box\vec{\exists}\Diamond(b$.info) means exactly the opposite of the previous statement, that is "button *info* never dies". This does not mean that it is always selectable, but rather that, in all situations, button *info* can eventually become selectable.

- $\vec{\exists}\Diamond$(b.option1 $\wedge$ b.option5) is interpreted as "there is a browsing session in which at some point buttons *option1* and *option5* are both selectable." This property may seems to require only a simple state-by-state check, looking for both out links, but for hypermedia systems that allow multiple content elements to be displayed concurrently (like Trellis) this is a non-trivial condition to check.

- $\vec{\forall}$(◇c.menu $\wedge$ c.menu$\mathcal{P}^{\Diamond}$c.identification) means "in all (forward) browsing sessions content element *menu* is eventually visible, but only sometime after content element *identification* is visible." Notice that the second conjunct only states "for all browsing sessions if *menu* is visible, then sometime in the past *identification* was visible."

- $\vec{\forall}$(c.no-help-mode$\mathcal{F}^{\Box}\neg$b.help) signifies "for all browsing sessions, if content element *no-help-mode* is visible then button *help* will never be selectable."

- $\vec{\forall}(\circ\psi$ **atnext** $\varphi)$ states that immediately after the *first* occurrence of φ, ψ will happen. Notice that the formula $\vec{\forall}(\varphi\mathcal{F}^{\circ}\psi)$ would be too strong since we would be requiring the original condition for *all* occurrences of φ, not just for the first one.

  Similarly: $\vec{\forall}(\psi$ **atnext** $\varphi)$ states that *during* the first occurrence of φ, ψ happens, and $\vec{\forall}(\psi$ **atnext** $\circ\varphi)$ says that *before* the first occurrence of φ, ψ must happen.

- $\overleftarrow{\forall}\Diamond_{=n}\varphi$ means "At this point of the browsing session, φ must have occurred exactly n times."

- $\vec{\forall}\Box\overleftarrow{\exists}\Box b$.help means "wherever you happen to be, you could have gotten there having the button *help* always selectable."

- $\vec{\exists}\Box\neg b$.confidential means "there is a way to browse the hypertext such that the button *confidential* is never selectable".

Model checking: Validating a subset of HTL*

HTL* has been presented as a rich temporal logic language especially suited for expressing browsing properties for hypermedia documents. We now describe a method for verifying whether particular HTL* formulae hold for specific hyperdocuments. This kind of problem has already been extensively studied in other contexts, most notably in the domain of concurrent programs and protocols specified by collections of finite state machines. These solutions are collectively referred to as *model checking*.

We will not present a lengthy technical discussion of model checking methods here; however, a brief description is in order. Our contributions are not in developing new checking algorithms but rather in showing how to use the program analysis work of Clarke [11] for analyzing hyperdocuments. Model checking algorithms traverse the finite state machine of a Kripke structure and verify (or disprove) the truth of a particular temporal logic formula; this involves searching down appropriate combinations of paths (according to the meanings of the temporal operators in the formula) and checking the truth of the individual state assertions at each node along the way.

In the general case, and especially with complicated temporal formulae, the complexity of model checking may require exponential time. This has been demonstrated to be the case for CTL* [11], and HTL* is semantically very similar. However, Emerson and Lei have shown that in most *practical* cases model checking of CTL* is tractable even for very large systems [14]. Based on previous results by Lichtenstein and Pnueli [26], Emerson and Lei determined that formulae of CTL* which combine both branching-time and linear-time operators could be checked with complexity exponential in the length of the temporal formula, yet *linear* in the size of the structure.

In practice, the HTL* formulae that hypertext authors would want to examine would not be overly complex, and the size of the model only affects the checking algorithm in a linear manner. For most hypermedia systems, generating a model from a hyperdocument is also of linear complexity. Therefore, we believe that model checking the entire HTL* language would be practical for most cases.

Clarke, *et al.*, have studied the model checking problem extensively for a restricted temporal logic language called CTL[11, 12, 5]. CTL is a proper subset of CTL* formed by requiring temporal operators to appear only in specific combinations. Clarke has implemented an efficient algorithm that checks CTL in polynomial time for *all* cases. The restrictions on operators limit the combinatorial explosion of paths when searching the finite state machine.

Rather than implementing a model checker for full HTL* we have chosen to experiment using Clarke's software. We correspondingly defined a proper subset of HTL* called HTL, semantically similar to CTL. HTL permits only branching-time operators, meaning each path quantifier must be immediately followed by exactly one of the operators $\circ$ or $\mathcal{U}$. The definition of HTL can be stated more formally:

- Every simple formula in the state language is a formula in HTL.
- If φ and ψ are HTL formulae, then so are $\neg\varphi$, $\varphi \vee \psi$, $\overrightarrow{\exists}\Diamond(\varphi)$, $\overleftarrow{\exists}\Diamond(\varphi)$, $\overrightarrow{\exists}[\varphi \mathcal{U} \psi]$, $\overleftarrow{\exists}[\varphi \mathcal{U} \psi]$.

For formal simplicity, the $\overrightarrow{\forall}$ operator and the path quantifier $\Box$ are defined in terms of $\overrightarrow{\exists}$ and $\Diamond$; practically, we write formulae containing those symbols as if they were technically part of HTL.

Because of the backward path operators, HTL turns out to be syntactically equivalent to POTL of Pinter and Wolper [34] rather than to CTL. As one can see in the formation rules, HTL

gives rise to an inseparable combination of temporal operators. For instance, $\overrightarrow{\exists}\circ$ and $\overleftarrow{\exists}\circ$ stand for "there is an immediate successor" and "there is an immediate predecessor" respectively. All the usual abbreviations are defined and interpreted in the obvious way. Although HTL severely restricts the expressiveness power of HTL* it is still appropriate to express many interesting properties in the context of hypertext.

Links-automata provide models for model checking

To verify browsing properties of a hyperdocument with the model checker, there must first be a model to check. The specific method of extracting a model from a hyperdocument will vary from system to system, but in general the idea is to use the links-automaton of a hyperdocument to produce an appropriate finite state machine for the Kripke structure required by the model checker. For many hypermedia systems this is a simple, if not trivial, exercise; for other systems, notably Trellis, converting the links-automaton to an FSM can be quite involved. There are, then, three issues in modeling a hyperdocument:

1. What is the appropriate links-automaton for the *hypermedia system* that generated the document?
2. How shall this links-automaton lead to a finite state machine?
3. What state assertions shall be associated with the FSM, i.e., what atomic properties of the hyperdocument characterize the aspects of browsing that are to be studied?

In the following two sections, we illustrate these processes for two different hypermedia systems: Trellis and Hyperties. For Hyperties (and most other hypermedia systems) the most appropriate links-automaton is already a finite state machine, so model generation is fairly simple. For Trellis the correct links-automaton is a PT-net, which is more powerful than a finite state machine; extracting a checkable model from a Trellis document is therefore a more complex process.

In our examples, typewriter font indicates direct input and output for Clarke's model checking software, which uses ASCII character representations for the CTL (HTL) operators. "A, E" correspond to "$\overrightarrow{\forall}$, $\overrightarrow{\exists}$"; "G, F, U, X" are "$\Box, \Diamond, \mathcal{U}, \circ$" respectively[3]. To aid in following this notational shift, we show some formulae in both symbolic operator form and in ASCII model checker input form.

3 Analyzing Trellis hyperdocuments

Trellis [40, 39, 41] is a unique hypermedia system developed previously by the authors. It differs from other systems in that links can have multiple source nodes and multiple destination nodes. This allows creation and synchronization of parallel browsing paths with multiple concurrently displayed content elements. The novel link concepts of Trellis have been most recently been realized in a WWW browser called MMM (Multi-head Multi-tail Mosaic) [8].

The Trellis document shown in figure 3 is a small net that expresses the browsing behavior found in some hypertext systems, namely that when a link is followed out of a node, the source content stays visible and the target content is added to the screen. The source must later be explicitly disposed of by clicking a *remove* button.

[3]CTL does not contain backwards path operators, so we have restricted our experiments with Clarke's model checker to forward path properties.

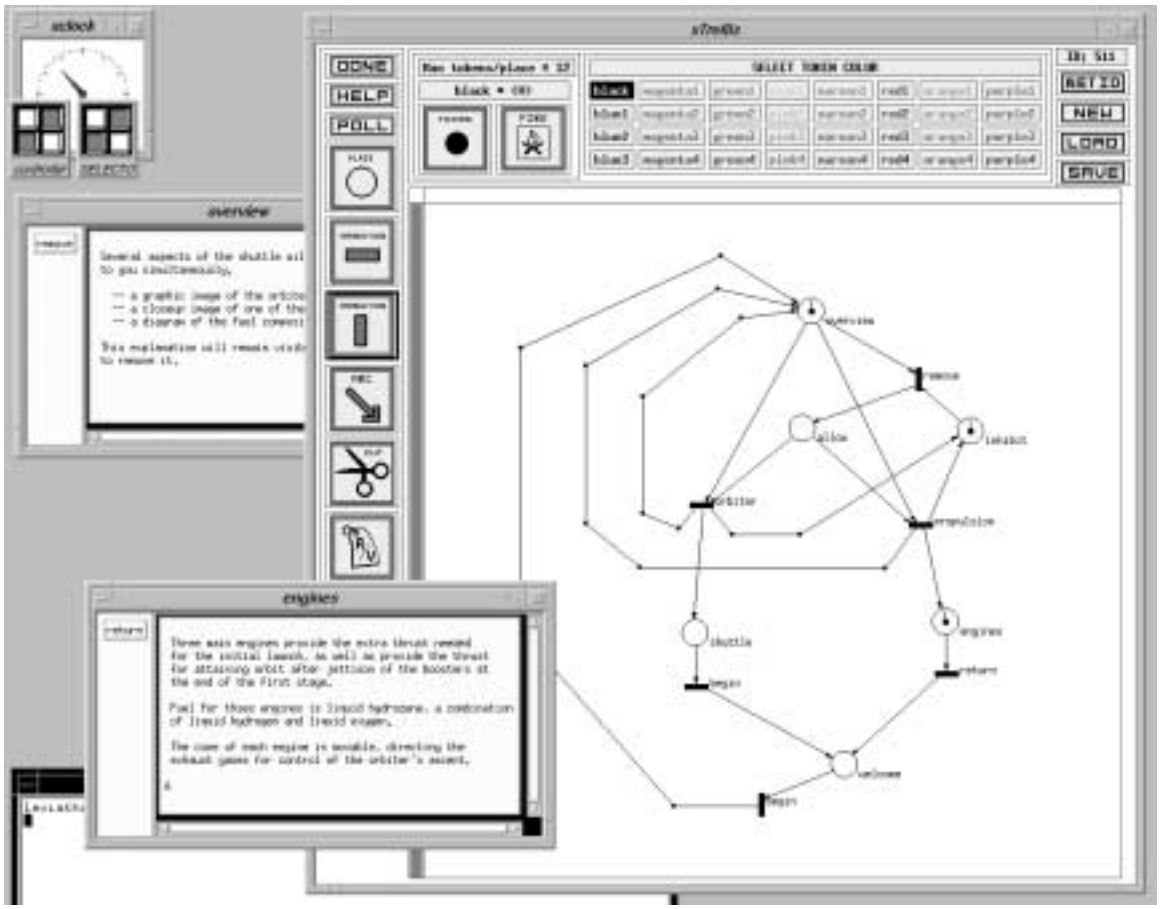

Figure 3: Small Trellis document.

The PT-net is the links-automaton of this hyperdocument; a PT-net is not always a finite state machine. In order to obtain a finite state machine for model checking, we compute the coverability graph of the net [33, pp. 91-95] and use that. The net here gives rise to a coverability graph of 11 states (reference [40] shows in detail how a coverability graph can be generated from the PT-net in a Trellis document). The coverability graph is in FSM form, but has some redundant states; after simplification, we have an 8-node FSM.

The transition diagram of this FSM and its encoding for the model checker are shown in Figure 4. The number of states and transitions (*CUBES*) is given first. Next, the section *MOORE-OUTPUTS* names the atomic predicates we are using[4]. The bit vector next to a state number indicates which of the atomic predicates (counted in the order listed in *MOORE-OUTPUTS*) are true in that state. Finally, the transitions out of a state are given as a list of destination state numbers following each bit vector.

Notice that in this particular document, several states have multiple concurrently visible

[4]Recall that for our experiment, a predicate like `c.welcome` means the *welcome* content element is visible, and `b.begin` means the *begin* link is selectable; we have not attempted to encode any document properties other than these simple ones, though more complicated properties could certainly be represented.

```
NAME = RefB.fsm;
INPUTS = ;
STATES = 8;
CUBES = 12;
MOORE-OUTPUTS =
  c.welcome,c.overview,c.shuttle,c.engines,c.allow,c.inhibit,
  b.begin,b.orbiter,b.propulsion,b.start,b.return,b.remove;

#0   100010100000
     1
#1   010010011000
     2
     6
#2   010101000011
     3
     4
#3   000110000010
     0
#4   110001100001
     0
     5
#5   010001000001
     1
#6   011001000101
     7
     4
#7   001010000100
     0
#END
```

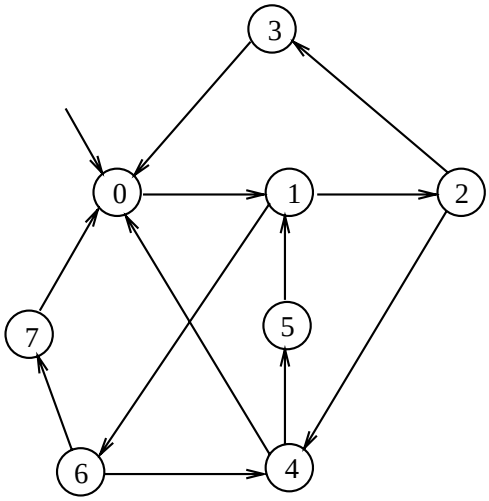

Figure 4: FSM encoding of the links-automaton for Trellis document

content elements. State 2, for example, has bits marked in positions 2 and 4, meaning the predicates *c.overview* and *c.engines* are true. This in turn implies that these two content elements are concurrently visible when browsing the hyperdocument at the point modeled by state 2. This is exactly the browsing situation illustrated in figure 3.

This model can be queried with the model checker for any browsing properties that involve temporal combinations of the atomic predicates. In our experiment, due to our choices for atomic predicates, we can investigate properties about when during browsing certain links will be selectable, or under what conditions certain content elements will be visible:

- Is there some browsing path such that at some point both the "orbiter" and "propulsion" buttons are selectable on one screen?
  $\vec{\exists}\Diamond$(b.orbiter $\wedge$ b.propulsion)

```
|= EF(b.orbiter & b.propulsion).
```

```
The formula is TRUE.
```

- Does there exist a browsing path such that at some point both the "shuttle" text and the "engines" text are concurrently visible?
  $\vec{\exists}\Diamond$(c.shuttle $\wedge$ c.engines)

```
|= EF(c.shuttle & c.engines).
The formula is FALSE.
```

- Is there some browsing path that reaches a point at which no buttons are selectable?

```
|= EF( ~b.begin & ~b.remove & ~b.propulsion & ~b.orbiter
       & ~b.start & ~b.return ).
The formula is FALSE.
```

- Is there some browsing path that will eventually simultaneously show the "overview" panel and the "engines" panel?

```
|= EF(c.overview & c.engines).
The formula is TRUE.
```

  An alternate way to express this query is to reverse the sense: On all paths is it always the case that either "overview" or "engines" is not showing?
  $\vec{\forall}\Box(\neg$c.overview$\vee\neg$c.engines$)$

```
|= AG(~c.overview | ~c.engines).
The formula is FALSE.
```

- Is it possible during browsing to see both the "welcome" and "engines" panels on the same screen?

```
|= EF(c.welcome & c.engines).
The formula is FALSE.
```

- Can both the "allow" access control and the "inhibit" access control ever be in force at the same time?

```
|= EF(c.inhibit & c.allow).
The formula is FALSE.
```

- Is it possible to select the "orbiter" button twice on some browsing path without selecting the "remove" button in between?
  $\vec{\exists}\Diamond$(b.orbiter $\wedge$ $\vec{\forall}\circ(\vec{\forall}$[b.remove $\mathcal{U}$ b.orbiter$]))$

```
|= EF(b.orbiter & AX(A[b.remove U b.orbiter])).
The formula is FALSE.
```

This last query is a bit more complex. The informal expression of it does not parallel closely the actual operators employed to check the property.

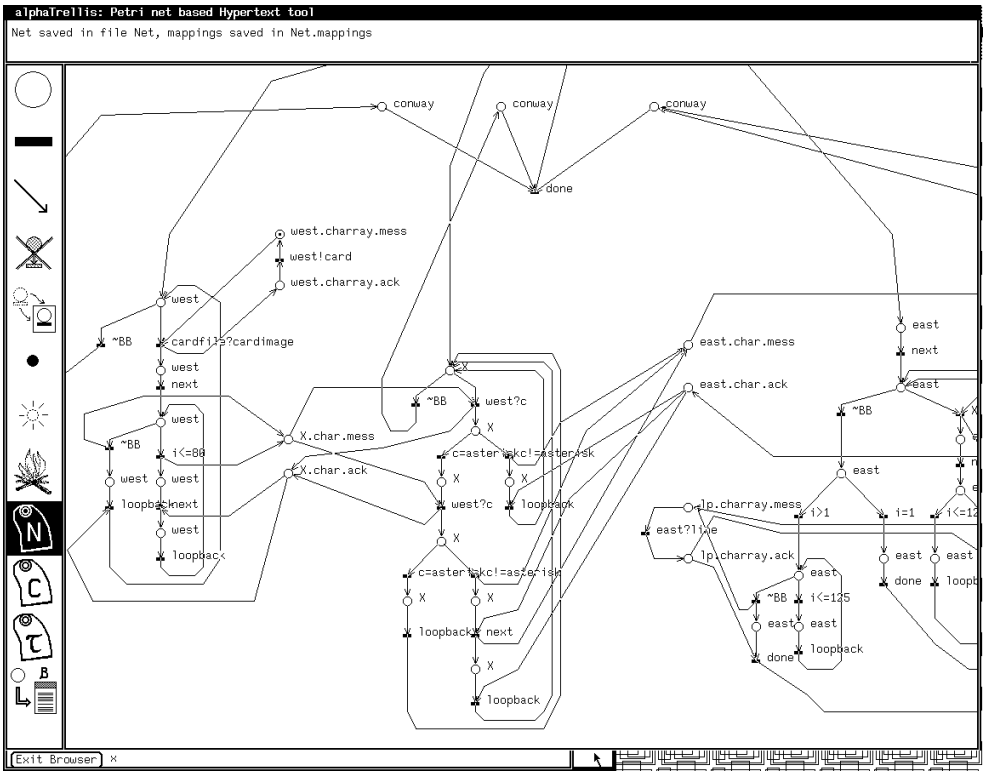

Figure 5: CSP program as browsable Trellis document

A larger Trellis document

The model checker also has been tested on the larger Trellis document shown in Figure 5 (being displayed by an older version of the Trellis net editor). The document contains about 70 nodes; the state machine derived from this net contains more than one thousand states. The performance of the model checker on formulae like those from the previous example was on the order of a few seconds, with the most complicated query we tried requiring about 20 seconds on an older-generation engineering workstation. We suspect that authors of hyperdocuments will find such timing not at all unreasonable for establishing the presence or absence of critical browsing properties.

Colored PT-net models for CSCW

Current work with Trellis involves encoding group collaboration protocols into hypermedia documents [17]. The links-automata on which these CSCW hyperdocuments are based is a colored PT-net. FSM models derived from colored PT-nets are even more complicated than those derived from plain PT-nets; the group interactions encoded in these models benefit greatly from the verification analysis enabled by model checking.

For example, consider the colored PT-net net shown in Figure 6. This work, as implemented in the WWW, is further discussed in two Web conference papers [8, 9], both of which can be found on the Web. We analyzed a Trellis hyperdocument based on this net with the model checker, and found an error in its behavior. The "hold" place was intended to allow the modera-

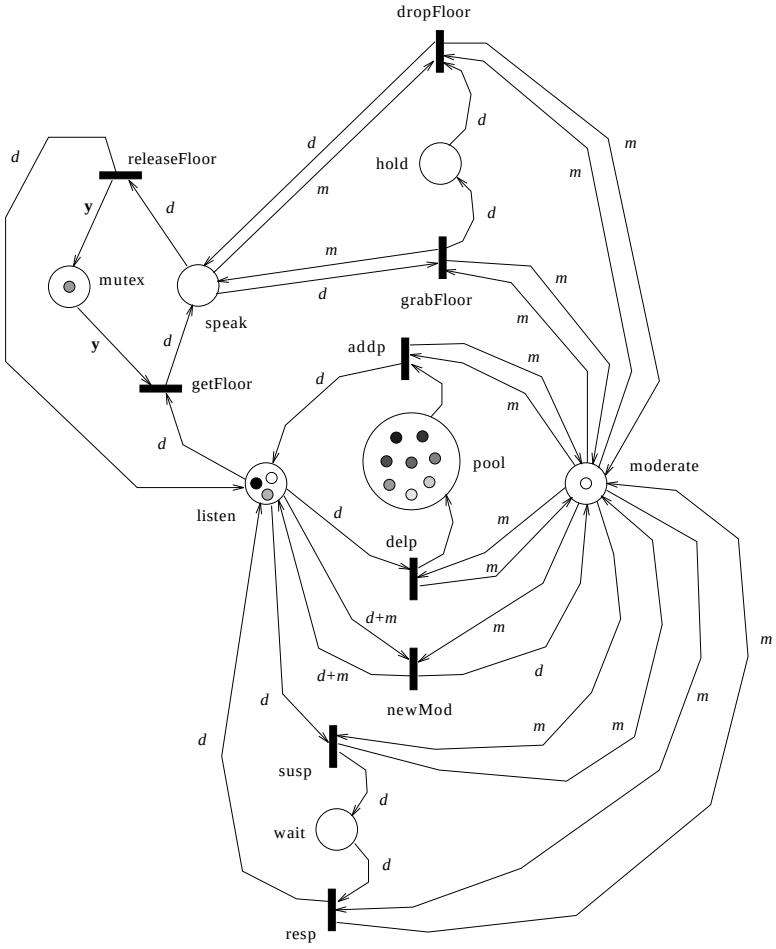

Figure 6: Colored PT-net for moderated meeting protocol

tor to park a speaker for a time while the moderator assumed the speaker's floor; the moderator would then release the floor back to the speaker on hold. The net as shown here, however, does not constrain the moderator to execute the "dropFloor" operation after the "grabFloor" operation. If, instead, a "grabFloor" is followed by a "releaseFloor" operation, the previous speaker is stuck on hold. To check for this problem, we asserted this formula:

```
EF ( c.hold & ~c.speak ) .
```

Informally, this says "There is some browsing path that reaches a document state in which the *hold* node is active (*i.e.*, someone is on hold) and the *speak* node is not active (*i.e.*, no one is speaking, moderator or otherwise)." The model checker revealed the assertion to be *true*, though we expected it to be *false*, thus uncovering an error in how the link structure of the net encodes the desired browsing behavior.

4 Analyzing a Hyperties document

To illustrate that these browsing verification concepts are not specific to Trellis, but are indeed general, we have applied the approach to a Hyperties [37, 30] document. Specifically, we have extracted the link structure from the Hyperties version of ACM's *Hypertext On Hypertext* issue of the *Communication of the ACM*, and use the directed graph as the finite state machine for the model checker. The model contains 298 states (content nodes), and over 700 links. Here are the results of some sample queries, all of which were answered by the model checker in less than two seconds on an engineering workstation.

- On all paths from the initial node, is it always the case (at all nodes) that some future path contains the *Table of Contents (TOC)* node?
  $\overrightarrow{\forall}\Box(\overrightarrow{\exists}\Diamond(\text{c.tableofc}))$

  ```
  |= AG ( EF (c.tableofc)).
  The formula is FALSE.
  ```

  Since this property does not hold, this says the document contains at least one node from which it is impossible to reach the *TOC*. Note that these properties are being tested for the link structure of the document alone, and do not account for any system-supplied browsing aids, such as backup or full history.

- On all paths from the initial node, will the *TOC* eventually be seen?

  On all paths from the initial node, will the *Introduction* eventually be seen?

  ```
  |= AF (c.tableofc).
  The formula is FALSE.

  |= AF (c.introduc).
  The formula is TRUE.
  ```

  The previous property asked if *TOC* could be reached from any node; this property is different in that it asks if *TOC* will eventually be reached (at least once) when browsing from the initial node. Such a property could be satisfied if, for example, *TOC* were the first or second node on all paths, whether or not it could then be returned to after browsing past it. The second formula above illustrates this with the *Introduction* node. Since the *Introduction* is the initial node of the document, the property trivially holds, but it says nothing about whether the *Introduction* can be revisited once it is passed.

- Is there at least one path such that once the *Introduction* is reached, it can be later revisited?
  $\overrightarrow{\exists}\Diamond(\text{c.introduc} \wedge \overrightarrow{\exists}\circ(\overrightarrow{\exists}\Diamond(\text{c.introduc})))$

  ```
  |= EF (c.introduc & EX (EF (c.introduc))).
  The formula is FALSE.
  ```

Since the *Introduction* is the first node on all paths, the failure of this property implies that it can be read only once during any traversal of the document (again, considering only the link structure proper). In Hyperties, this means the special browser features, like history, must be relied on.

- On all paths from the initial node, is it always the case (at all nodes) that the *TOC* is exactly one link away?

  On all paths from the initial node, is it always the case (at all nodes) that the *TOC* is no more than three links away?

```
|= AG (EX (c.tableofc)).
The formula is FALSE.

|= AG (EX (c.tableofc | EX (c.tableofc | EX (c.tableofc) ) ) ).
The formula is FALSE.
```

  This first property might well be one an author wants a document to have, though it is a bit strong. The second property is more relaxed and more probable for a document to exhibit. The proper semantics are obtained by nesting the "next step" operator. Of course, neither property holds for this document since we have already seen that some nodes exist from which *TOC* is not reachable at all.

- Is there some path from the initial node on which it is always the case (at all nodes) that the *TOC* is exactly one link away?

```
|= EG (EX (c.tableofc)).
The formula is TRUE.
```

  Interestingly enough, though we saw that the "one hop away" property for *TOC* does not hold for all paths, it does hold for at least one path in this document.

- On all paths from the initial node, is it always the case (at all nodes) that either the *Bibliography* is not displayed, or else some path eventually leads from it to the *TOC*?

```
|= AG(~c.bibliogr | EF(c.tableofc)).
The formula is FALSE.
```

  This query attempts to further refine the finding that the *TOC* cannot be reached from some nodes. Here we see that the *Bibliography* node is one such node.

5 Discussion and conclusions

In this report we have described an authoring mechanism whereby the creator of a hyperdocument can verify that its link structure will exhibit certain desired properties when browsed. This method is automatable; author input is required to express the properties to be checked, but no human-generated proof is required.

Modeling other hypermedia systems

We illustrated model checking for two hypertext systems: Trellis, and Hyperties, but our analysis methods will likewise apply to documents from any hypermedia system for which an accurate model can be derived. Producing a model for analysis requires first determining how much of the system's browser behavior to add to the document links-automaton, based on the purpose for an analysis.

In our experiments we worked directly with the links-automaton without special browser features mixed into the model. Hyperties, for example, allows a reader to jump from any node directly to the *Table of Contents* node; this behavior is maintained by the browser software, and is not represented directly as links in the document. Correspondingly, the model we analyzed (the *CACM* issue) gave the answer *false* to the query "*on all paths, the TOC is eventually seen*". In this document, the author relied on the special browser feature and built a linked structure that did not always allow return to the *Table of Contents*. If browsed on a system other than Hyperties, one without this special feature, this particular document structure would lead readers down dead ends.

The choice to analyze links-automaton behavior alone was arbitrary; browser behavior can certainly be encoded into a model if desired. The Hyperties *Table of Contents* feature just mentioned can be included in analysis by placing extra transitions into the FSM processed by the model checker.

Path mechanisms and scripting languages

Though the method applies to general hyperdocuments, it also has application to hypermedia/multimedia scripting languages and other path control mechanisms. In fact, the Trellis model we used in some of the examples is inherently such a path control mechanism, providing the author of a hyperdocument greater control over the sequences in which nodes will be browsed; this control is obtained through a more powerful links-automaton (the PT-net) than that used in other hypermedia systems (finite state machines). Other researchers have described different mechanisms to meet similar needs [10, 44]. Our model checking approach could be used, for example, during document maintenance to verify that paths expressed in these other mechanisms continue to be accurate for the target hyperdocuments after modification of the link structure or contents.

User interface

Model checking as we have presented it would benefit greatly from a good user interface and support environment. Hyperdocument authors will almost certainly not wish to write browsing properties directly in the HTL* notation. While some success might be had in providing a more English-like notation that could be translated directly into HTL* a better approach might be to pre-design dozens, or perhaps hundreds, of queries that make sense for general hyperdocuments. These would be pre-coded into HTL* and provided in a menu for authors to check as desired. New queries could be added to the menu as authors decided to write them; we expect that writing correct queries in HTL* would be about as difficult as writing macros for document-processing tools like TEX or *emacs*.

System interoperability

Finally, the concept of a links-automaton provides an intriguing solution to the interoperability problem. The current approach, seen in HTML and the World Wide Web, is *heterogeneity through homogeneity* — standardizing on a common document notation and a (usually large) collection of specialized browsing operations and interpretations. An alternate approach is to have each hypermedia system export a links-automaton for a document, expressing the behavior inherent in the links of each document without all the "extra" features provided by the individual browser software. Any browser that could "execute" the finite state automaton would provide the intended behavior for the document; moreover, the simplicity of the FSM model means that all browsers could provide a "common" behavior for any particular document.

References

[1] Atlee, J. M., and Gannon, J. D. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering* (Jan. 1993), 24–40.

[2] Barringer, H., Kuiper, R., and Pnueli, A. Now you may compose temporal logic specifications. In *Proc. of the Sixteenth ACM Symp. on Theory of Computing* (1984), pp. 51–63.

[3] Beeri, C., and Kornatzky, Y. A logical query language for hypertext systems. In *Hypertext: Concepts, Systems, and Applications*, A. Rizk, N. Streitz, and J. André, Eds. Cambridge University Press, Nov. 1990, pp. 67–80. Proceedings of the European Conference on Hypertext.

[4] Ben-Ari, M., Pnueli, A., and Manna, Z. The temporal logic of branching time. In *fProc. Eighth ACM Symp. on Principles of Programming Languages* (1981), pp. 164–176.

[5] Bursh, J. R., Clark, E. M., and McMillan, K. L. Symbolic model checking: 10 to the 20 states and beyond. Carnegie Mellon and Stanford Universities, 1989.

[6] Burstall, R. M. Program proving as hand simulation with a little induction. *Information Processing 74* (1974), 308–312.

[7] Campbell, B., and Goodman, J. M. HAM: A general purpose hypertext abstract machine. *Commun. ACM 31*, 7 (July 1988), 856–861.

[8] Capps, M., Ladd, B., and Stotts, D. Enhanced graph models in the web: Multi-client, multi-head, multi-tail browsing. In *Proceedings of the 5th WWW Conference (Computer Networks and ISDN Systems, vol. 28)* (May 6-10 1996), pp. 1105–1112. This paper appears on-line at http://www5conf.inria.fr/fich_html/papers/P19/Overview.html.

[9] Capps, M., Ladd, B., Stotts, D., and Furuta, R. Multi-head/multi-tail mosaic: Adding parallel automata semantics to the web. In *Proceedings of the 4th WWW Conference (WWW Journal, O'Reilly and Associates Inc., vol. 1)* (Dec. 11-14 1995), pp. 433–440. This paper appears on-line at http://www.w3.org/pub/Conferences/WWW4/Papers/118/.

[10] Christodoulakis, S., Theodoridou, M., Ho, F., and Papa, M. Multimedia document presentation, information extraction, and document formation in MINOS: A model and a system. *ACM Trans. Office Inf. Syst. 4*, 4 (Oct. 1986), 345–383.

[11] Clarke, E. M., Emerson, E. A., and Sistla, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems 8*, 2 (1986), 244–263.

[12] Clarke, E. M., and Grumberg, O. Research on automatic verification of finite-state concurrent systems. *Ann. Rev. Comput. Sci. 2* (1987), 269–290.

[13] Emerson, E. A., and Halpern, J. Y. "sometimes" and "not never" revisited: on branching vs. linear time. In *Proc. Tenth ACM Symp. on Principles of Programming Languages* (1983), pp. 127–140.

[14] Emerson, E. A., and Lei, C. L. Modalities for model checking: Branching time strikes back. In *Twelfth Symposium on Principles of Programming Languages* (1985), pp. 84–96.

[15] Emerson, E. A., and Srinivasan. Branching time temporal logic. In *Proc. of the REX School/Workshop 1988 on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency* (1989), Springer-Verlag.

[16] Furuta, R., and Stotts, P. D. Structured dynamic behavior in hypertext. Technical Report CS-TR-2597 (UMIACS-TR-91-14), University of Maryland Department of Computer Science and Institute for Advanced Computer Studies, Jan. 1991.

[17] Furuta, R., and Stotts, P. D. Interpreted collaboration protocols and their use in groupware prototyping. In *Proc. of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW '94)* (Oct. 1994), ACM, New York, pp. 121–131.

[18] Gabbay, D., Pnueli, A., Shelah, S., and Stavi, J. On the temporal analysis of fairness. In *Proc. Seventh ACM Symp. on Principles of Programming Languages* (1980), pp. 163–173.

[19] Galton, A., Ed. *Temporal Logics and their applications*. Academic Press, Harcourt Brace Jovanovich, Publishers, 1987.

[20] Halasz, F. G. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Commun. ACM 31*, 7 (July 1988), 836–852.

[21] Kahn, K., and Gorry, G. A. Mechanizing temporal knowledge. *Artificial Intelligence 9* (1977), 67–95.

[22] Kowalski, R. A., and Sergot, M. J. A logic-based calculus of events. *New Generation Computing 4* (1983), 67–95.

[23] Kroger, F. *Temporal Logic of Programs*. Springer-Verlag, 1987.

[24] Lamport, L. "Sometime" is sometimes "not never": on the temporal logic of programs. In *Proc. Seventh ACM Symp. on Programming Languages* (1980), pp. 174–185.

[25] Lee, R. M., Coelho, H., and Cotta, J. C. Temporal inferencing on administrative databases. *Information Systems 10* (1985), 197–206.

[26] Lichtenstein, O., and Pnueli, A. Checking that finite state concurrent programs satisfy their linear specification. In *Twelfth Annu. ACM Symp. Principles on Programming Languages.* (1985), pp. 97–107.

[27] Lloyd, J. W. *Foundations of Logic Programming*, second ed. Springer-Verlag, 1987.

[28] Manna, Z. *Temporal Verification of Reactive Systems*. Springer-Verlag, 1995.

[29] Manna, Z., and Pnueli, A. Verification of concurrent programs: the temporal framework. In *The Correctness Problem in Computer Science*, R. S. Boyer and J. S. Moore, Eds. Academic Press, 1981, pp. 215–273.

[30] Marchionini, G., and Shneiderman, B. Finding facts vs. browsing knowledge in hypertext systems. *Computer 21*, 1 (Jan. 1988), 70–80.

[31] Moszkowski, B. *Reasoning about digital circuits*. PhD thesis, Standford University, CA, 1983.

[32] Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*, 4 (Apr. 1989), 541–580.

[33] Peterson, J. L. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Inc., 1981.

[34] Pinter, S. S., and Wolper, P. A temporal logic for reasoning about partially ordered computations. In *Proc. of the third ACM Symp. on Principles of Distributed Computing* (1984).

[35] Pnueli, A. The temporal logic of programs. In *Proc. Eighteenth IEEE Symp. on Foundations of Computer Science* (1977), pp. 46–67.

[36] Reisig, W. *Petri Nets: An Introduction*. Springer-Verlag, 1985.

[37] Shneiderman, B. User interface design for the Hyperties electronic encyclopedia. In *Proceedings of Hypertext '87* (Nov. 1987), pp. 189–194. Published by the Association for Computing Machinery, 1989.

[38] Sinachopoulos, A. Logics for Petri-nets: Partial order logics, branching time logics and how to distinguish between them. *Petri Net Newsletter* (8 1989), 9–14.

[39] Stotts, D., and Furuta, R. Browsing parallel process networks. *Journal of Parallel and Distributed Computing 9*, 2 (1990), 224–235.

[40] Stotts, P. D., and Furuta, R. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems 7*, 1 (Jan. 1989), 3–29.

[41] Stotts, P. D., and Furuta, R. Temporal hyperprogramming. *Journal of Visual Languages and Computing 1*, 3 (1990), 237–253.

[42] Stotts, P. D., Furuta, R., and Ruiz, J. C. Hyperdocuments as automata: Trace-based browsing property verification. In *Proceedings of the ACM Conference on Hypertext (ECHT '92)*, D. Lucarella, J. Nanard, M. Nanard, and P. Paolini, Eds. ACM Press, 1992, pp. 272–281.

[43] Wolper, Z. Temporal logic can be more expressive. In *Proc. of the Twenty-second IEEE Symp. on Foundations of Computer Science* (1981), pp. 340–348.

[44] Zellweger, P. T. Active paths through multimedia documents. In *Document Manipulation and Typography*, J. C. van Vliet, Ed. Cambridge University Press, Apr. 1988, pp. 19–34. Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography, Nice (France), April 20–22, 1988.

Appendix A

A.1 Syntax of HTL*

The kernel of HTL* is the language we call SL (State Language), a typed first-order language [27]. Typed first-order languages, also known as many-sorted languages, are more convenient than ordinary first-order logic for expressing properties of systems with different types of primitive elements (e.g. buttons/links and content elements). SL allows description of the properties of hyperdocuments at particular points during browsing. To keep early experiments simple, the SL in this report allows expression of only two such properties: "is a content element visible?", and "is a link (button) selectable?". For example, the SL formula *c.hello* asserts that the content element named "hello" is visible; similarly, the formula *b.help* asserts that the button[5] named "help" is selectable. Practical use will require augmenting the expressive power of SL by introducing new atomic predicates to encode other aspects of hypermedia documents and browsing.

HTL* is a temporal extension of SL; it includes all symbols of SL with additional temporal logic operators, including backward path quantifiers (as explained later), using a notation similar to that of Wolper [43] and Lamport [24]. Except for the difference in the kernel and the backward path quantifier, HTL* is essentially the same language as CTL*.

Well-formed formulae

Let a *path* be a sequence of states. We define inductively two kinds of formula: *state formulae* describing what is the case at a specific state, and *path formulae* describing what holds over an entire path. The well formed HTL* formulae are then exactly the *finite state formulae* generated by these rules:

- State formulae:

  S1. All atomic formulae of SL are state formulae; we refer to these as *simple* formulae.

  S2. If φ and ψ are *state* formulae then so are $\neg(\varphi)$, $(\varphi \vee \psi)$.

  S3. If φ is a *path* formula then $\overset{\rightarrow}{\exists}(\varphi)$ and $\overset{\leftarrow}{\exists}(\varphi)$ are *state* formulae.

- Path formulae:

  P1. All *state* formulae are path formulae.

  P2. If φ and ψ are path formulae then so are $\neg(\varphi)$, $(\varphi \vee \psi)$.

  P3. If φ and ψ are path formulae then so are $(\varphi \mathcal{U} \psi)$ and $\circ(\varphi)$.

The unary operator $\circ$ (next time), and the binary operator $\mathcal{U}$ (until) are termed *tense operators*; they are used to formulate assertions over sequences of states (paths). The formula $\varphi \mathcal{U} \psi$ holds for a path if and only if ψ eventually occurs in the sequence, and φ is true in all states from the beginning *until* the first state where ψ is true. Nothing else is implied; that is, φ may or may not continue to hold thereafter. Similarly, $\circ\varphi$ holds for a path if and only if φ is true in the *next* state in the path, that is, in the second state of the of the sequence.

[5] For historical reasons, we use the term "button" almost interchangeably with "link" and "anchor."

The symbols $\overrightarrow{\exists}$ and $\overleftarrow{\exists}$ mean "there is at least one *forward* (respectively *backward*) path from this state such that ...". These operators are called *existential path quantifiers*.[6]

Operator abbreviations

The symbols and operators given as defining HTL* comprise a *minimal* (but clearly not unique) set; all the other usual temporal operators can be constructed from them. We now define these, and other notations, as abbreviations for expressive convenience in describing properties of hyperdocuments and browsing.

- Standard abbreviations:
  - The logic symbols $\wedge$ (conjunction), $\rightarrow$ (conditional) and $\leftrightarrow$ (if and only if) are defined in the usual way. For instance, using DeMorgan's rules, we get for the symbol $\wedge$:
$$\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$$
  - The two unary tense operators $\Box$ (always) and $\Diamond$ (sometime) are defined as:
$$\Diamond(\varphi) \equiv \mathbf{true}\,\mathcal{U}\ \varphi$$
$$\Box(\varphi) \equiv \neg(\Diamond(\neg\varphi))$$
  - (φ **atnext** ψ), meaning "φ holds at the next state that ψ holds", is defined as:
$$\varphi\ \mathbf{atnext}\ \psi \equiv \neg\psi\mathcal{U}(\varphi\wedge\psi) \vee \circ\Box\neg\psi$$
  - (φ **unless** ψ), meaning "if there is a following state at which ψ holds then φ holds up to that point or else φ is always true", is defined as:
$$\varphi\ \mathbf{unless}\ \psi \equiv \varphi\mathcal{U}\psi \vee \Box\varphi$$
  - The universal path quantifiers $\overrightarrow{\forall}$ and $\overleftarrow{\forall}$ are defined as:
$$\overrightarrow{\forall}(\varphi) \equiv \neg(\overrightarrow{\exists}(\neg\varphi))$$
$$\overleftarrow{\forall}(\varphi) \equiv \neg(\overleftarrow{\exists}(\neg\varphi))$$
- Past tense operators: The 'past' operators: $\mathcal{P}^{\Box}$ (always in the past), $\mathcal{P}^{\Diamond}$ (sometime in the past) and $\mathcal{P}^{\circ}$ (immediately before). All of these operators have a 'conditional spirit', so to speak. For instance, $\varphi\mathcal{P}^{\Diamond}\psi$ is a *path* formula intuitively stating: " **if** φ holds in a certain state of the sequence, **then** sometime in the 'past' (i.e. in a previous state) ψ must have held." Notice that the eventual realization of φ is *not* assumed. The other two operators have the obvious similar intuitive meaning. The 'current' state is not considered as a 'past' state in our definitions. Formally we have:[7]
$$\varphi\mathcal{P}^{\Box}\psi \equiv \Box(\Box\neg\varphi \vee \psi\mathcal{U}\varphi)$$
$$\varphi\mathcal{P}^{\Diamond}\psi \equiv \Box\neg\varphi \vee \neg\varphi\mathcal{U}\psi$$
$$\varphi\mathcal{P}^{\circ}\psi \equiv \Box(\circ\varphi\rightarrow\psi)$$

[6] The backward path quantifier, as explained in section 5, refers specifically to the initial state. This is in some sense more restrictive than the backward operators of POTL, but it is all that is needed in this context. Moreover, this restriction will eventually simplify the checking process.

[7] The definition of $\mathcal{P}^{\Box}$ turns out to be similar to the definition of $\mathcal{U}$ in [43] where there is no "eventually" component. This operator is also known as the *unless* operator.

- FUTURE TENSE OPERATORS: The future operators: $\mathcal{F}^{\Box}$ (always in the future), $\mathcal{F}^{\Diamond}$ (sometime in the future) and $\mathcal{F}^{\circ}$ (immediately after) have analogous interpretations:

$$\varphi \mathcal{F}^{\Box} \psi \equiv \Box(\varphi \rightarrow \circ \Box \psi)$$

$$\varphi \mathcal{F}^{\Diamond} \psi \equiv \Box(\varphi \rightarrow \circ \Diamond \psi)$$

$$\varphi \mathcal{F}^{\circ} \psi \equiv \Box(\varphi \rightarrow \circ \psi)$$

EXAMPLES OF WFFS WITH ABBREVIATIONS

To illustrate the syntax of HTL*, we give a few examples of well-formed temporal formulae. Many of them are classical in temporal logic literature and are intended to be a quick exposition for readers not familiar with temporal logic. These are preliminary examples in the sense that the formal semantics of HTL* are not introduced until section 5.

- The formula $\Diamond\Box\varphi$ holds for a path if and only if there is a state in the sequence such that from this state on φ is always valid.

  Considering now the formula $\Box\Diamond\varphi$ we see that it has a different meaning altogether. It holds for sequence if and only if at each state of the sequence φ holds sometime in the future. If the sequence is infinite for example, then the previous statement amounts to saying that φ holds in infinitely many states in the sequence[8] or, in other words, that φ occurs infinitely often (also known as "fairness requirement").

  Correspondingly, the other combinations $\Box\circ\psi$, $\circ\Diamond\varphi, \ldots$ express different conditions.

- The following formulae may be considered an "unrolling" of the basic tense operators using the next time operator:

$$\Box\varphi \equiv \varphi \wedge \circ\Box\varphi$$

$$\Diamond\varphi \equiv \varphi \vee \circ\Diamond\varphi$$

$$\varphi \mathcal{U} \psi \equiv \psi \vee (\varphi \wedge \circ(\varphi \mathcal{U} \psi)) \wedge \Diamond\psi$$

- The following HTL* formulae *fail* to capture the intended meaning of the operator $\mathcal{F}^{\Box}$ (it is a good exercise to convince oneself of this):

$$\Diamond\varphi \rightarrow \Box\psi$$

$$\Diamond\varphi \rightarrow \circ\Box\psi$$

$$\Diamond\varphi \rightarrow \Diamond\varphi \mathcal{U} \circ\Box\psi$$

$$\Box(\Diamond\varphi \rightarrow \circ\Box\psi)$$

$$\Box(\varphi \rightarrow (\psi \wedge \circ\Box\psi))$$[9]

These are just a few examples. A complete list of temporal logical laws may be found in [23].

[8] Suppose not, then there is a "last" state in the sequence for which φ holds. But then in the next state (since the sequence is infinite) it would be false that sometime φ will hold; this yields a contradiction.

[9] Even though this formula does not capture the intented meaning, it provides a very similar one; the only difference is that the ψ must be true in all future states as well as in the current one.

A.2 Semantics of HTL*

Assume a hyperdocument H, containing nodes (with named content) and links (also with names). The semantics of HTL* are defined in terms of a Kripke structure K (an annotated finite state machine, as described in [11]) derived from H. Using a Kripke structure will allow the use of Clarke's model checker for determining the truth of HTL* statements, as explained in section 2. We also leave to section 2 any discussion of how a model is generated from H.

Let $K =< M, C, B >$ be a finite state machine and two functions. Let C be a function that maps names to the states in M. Let B be a function that maps names to the transitions in M. In SL an assertion "c.hello" is true for some state s if $C(s) = hello$, that is, if the name of the content mapped to s is "hello"; we are assuming that if the M is in state s, then the content for s is visible, or active. Similarly the SL assertion "b.help" is true in some state s if there is a transition t out of s such that $B(t) = help$; when M is in state s, the enabled transitions out of s represent selectable links in the hyperdocument being modeled by K.

Next, we have the concept of forward and backward *paths* or state sequences:

- $\sigma = (\rho_0, \rho_1, \rho_2, \ldots)$ is a forward sequence from state ρ_0 if and only if $\forall i, \rho_{i+1}$ is immediately reachable from ρ_i.
- $\sigma = (\rho_0, \rho_1, \rho_2, \ldots, M_0)$ is a backward sequence from state ρ_0 if and only if $\forall i, \rho_i$ is immediately reachable from ρ_{i+1}, and M_0 is the initial state of M.

We write first(σ) to refer to the state ρ_0, and σ_i to refer to the postfix of σ beginning with ρ_i; for instance $\sigma_2 = (\rho_2, \rho_3, \ldots)$. Notice that forward as well as backward sequences may be infinite, since paths may have cycles. This will be important for model checking issues.

Validity of HTL* formulae

It is clear that the validity of atomic formulae of SL on states in M can be completely determined using just H. That is to say, given a certain state, we can determine whether or not a link is selectable, or a particular content element is active in the modeled hyperdocument.

To define validity of more complex formulae we use the standard notation. Let M_0 be the initial state of M and ρ a reachable state from it. If φ is a *state* formula, we write:

$$M_0, \rho \models \varphi$$

to mean that formula φ is valid in the state ρ. A similar notation applies for *path* formulae holding for a sequence σ. When the initial marking M_0 is implicit then we omit it from the left hand side of the expression. The relation $\models$ is defined inductively:

1. Let φ be a simple formula. $\rho \models \varphi$ if and only if φ holds in state ρ. Similarly, $\sigma \models \varphi$ if and only if φ holds in state first(σ).
2. Let α be a path formula and ρ a particular state. The *path quantifiers* are defined as:
   - $\rho \models \overrightarrow{\forall}(\alpha)$ if and only if for every *forward* sequence σ from ρ (i.e. first(σ) = ρ), we have $\sigma \models \alpha$.
   - $\rho \models \overrightarrow{\exists}(\alpha)$ if and only if there is a *forward* sequence σ from ρ such that $\sigma \models \alpha$.
   - $\rho \models \overleftarrow{\forall}(\alpha)$ if and only if for every *backward* sequence σ from ρ (i.e. first(σ) = ρ), we have $\sigma \models \alpha$.

- $\rho \models \overleftarrow{\exists}(\alpha)$ if and only if there is a *backward* sequence σ from ρ such that $\sigma \models \alpha$.

3. Let φ and ψ be *state* formulae and σ a sequence. The *tense operators* are interpreted as:

   - $\sigma \models \Box(\varphi)$ if and only if $\forall i \geq 0,\ \text{first}(\sigma_i) \models \varphi$. That is to say, formula φ is valid in *all* states of the sequence σ.
   - $\sigma \models \Diamond(\varphi)$ if and only if $\exists i \geq 0,\ \text{first}(\sigma_i) \models \varphi$. That is to say, formula φ is valid in *at least one* state of the sequence σ.
   - $\sigma \models \circ(\varphi)$ if and only if $\text{first}(\sigma_1) \models \varphi$. That is, the formula φ is valid in the second state of σ.
   - $\sigma \models \varphi \mathcal{F}^{\Box} \psi$ if and only if

     $$(\exists i \geq 0,\ \text{first}(\sigma_i) \models \varphi) \rightarrow (\forall j,\ j > i \rightarrow \text{first}(\sigma_j) \models \psi)$$

     In other words, in σ, *if* φ holds in some state then ψ will hold in *all* the following states. The binary operators $\mathcal{F}^{\Diamond}$ and $\mathcal{F}^{\circ}$ are defined analogously.
   - $\sigma \models \varphi \mathcal{P}^{\Box} \psi$ if and only if

     $$(\exists i \geq 0,\ \text{first}(\sigma_i) \models \varphi) \rightarrow (\forall j,\ 0 \leq j < i \rightarrow \text{first}(\sigma_j) \models \psi)$$

     In other words, in σ *if* φ holds in some state then ψ must have happened in all the previous states. The operators $\mathcal{P}^{\Diamond}$ and $\mathcal{P}^{\circ}$ are defined analogously.
   - The operator $\mathcal{U}$ is given the standard interpretation[10]: $\sigma \models \varphi \mathcal{U} \psi$ if and only if

     $$\exists i \geq 0,\ (\text{first}(\sigma_i) \models \psi \wedge (\forall j,\ 0 \leq j < i \rightarrow \text{first}(\sigma_j) \models \varphi))$$

   - $\sigma \models \varphi$ **atnext** ψ if and only if

     $$(\exists i \geq 0,\ \text{first}(\sigma_i) \models \psi) \rightarrow (\text{first}(\sigma_j) \models \varphi \text{ for the } \textit{least } j \geq 0 \text{ such that } \text{first}(\sigma_j) \models \psi)$$

4. Let α and β be *path* formulae. The *tense operators* are interpreted as:

   - $\sigma \models \Box(\alpha)$ if and only if $\forall i \geq 0\ (\sigma_i \models \alpha)$.
   - $\sigma \models \Diamond(\alpha)$ if and only if $\exists i \geq 0\ (\sigma_i \models \alpha)$.
   - $\sigma \models \circ(\alpha)$ if and only if $\sigma_1 \models \alpha$.
   - $\sigma \models (\alpha \mathcal{F}^{\Box} \beta)$ if and only if

     $$(\exists i \geq 0, \sigma_i \models \alpha) \rightarrow (\forall j,\ j > i \rightarrow \sigma_j \models \beta)$$

     The binary operators $\mathcal{F}^{\Diamond}$ and $\mathcal{F}^{\circ}$ are defined similarly.
   - $\sigma \models \alpha \mathcal{P}^{\Box} \beta$ if and only if

     $$(\exists i \geq 0, \sigma_i \models \alpha) \rightarrow (\forall j,\ 0 \leq j < i \rightarrow \sigma_j \models \beta)$$

     The operators $\mathcal{P}^{\Diamond}$ and $\mathcal{P}^{\circ}$ are defined analogously.
   - $\sigma \models \alpha \mathcal{U} \beta$ if and only if $\exists i \geq 0, (\sigma_i \models \beta \wedge (\forall j,\ 0 \leq j < i \rightarrow \sigma_j \models \alpha))$.
   - $\sigma \models \alpha$ **atnext** β if and only if

     $$(\exists i \geq 0,\ \sigma_i \models \beta) \rightarrow (\sigma_j \models \alpha \text{ for the } \textit{least } j \geq 0 \text{ such that } \sigma_j \models \beta)$$

[10] Actually this interpretation implies $\varphi \mathcal{U} \psi \rightarrow \Diamond \psi$. However, some authors, like Wolper, give a slightly different interpretation to the operator $\mathcal{U}$ in such a way that ψ does not necessarily hold in the sequence. This is precisely the meaning of the operator **unless**.

A.3 Further HTL* abbreviations and extensions

Before presenting the examples, we develop additional notation. Let φ be a state formula and σ a path. We say that $\Diamond_n\varphi$ holds for path σ if and only if φ holds *at least* n times in the path σ. It is easy to construct formulae expressing $\Diamond_n$, for every n. For instance

$$\Diamond_3\varphi \;\equiv\; \Diamond\varphi \wedge (\varphi \rightarrow (\Diamond\varphi \wedge \varphi \rightarrow \Diamond\varphi))$$

In general, we inductively define the formulae $\Diamond_n\varphi$ (for any natural n) as:

$$\Diamond_n\varphi = \begin{cases} \Diamond\varphi & \text{if } n = 1 \\ \Diamond\varphi \wedge (\varphi \rightarrow \Diamond_{n-1}\varphi) & \text{if } n > 1 \end{cases}$$

From this definition, one can immediately express two concepts:

- φ happens *at most* n times $\equiv\; \neg\Diamond_{n+1}\varphi$.
- φ happens *exactly* n times $\equiv\; \Diamond_n\varphi \wedge \neg\Diamond_{n+1}\varphi$.

We write $\Diamond_{\leq n}\varphi$, $\Diamond_{\geq n}\varphi$, and $\Diamond_{=n}\varphi$ to mean that φ happens at most, at least, or exactly n times respectively. By boolean combinations of the previous formulae one can express the fact that a property hold between n and m times, and so on.

Extensions of the $\mathcal{U}$ operator are also possible. We write $\varphi\,\mathcal{U}_n\psi$ to mean that φ occurs in all states of the sequence until the n$^{\text{th}}$ occurrence of ψ. As in the case of $\mathcal{U}$, nothing else is implied in the definition. That is, φ may or may not continue to hold thereafter. This concept is easily defined by recursion:

$$\varphi\mathcal{U}_n\psi = \begin{cases} \varphi\mathcal{U}\psi & \text{if } n = 1 \\ \varphi\mathcal{U}(\psi \wedge (\varphi\mathcal{U}_{n-1}\psi) & \text{if } n > 1 \end{cases}$$

For instance $\varphi\mathcal{U}_3\psi \;\equiv\; \varphi\mathcal{U}(\psi \wedge \psi\mathcal{U}(\psi \wedge \varphi\mathcal{U}\psi))$.

The extensions of the $\circ$ (next time) operator are similar. We write $\circ_{\leq n}\varphi$ to mean that φ will hold in all the states of the sequence up to, and including, the n$^{\text{th}}$ state. We write $\circ_{=n}\varphi$ to mean that φ will hold in the n$^{\text{th}}$ state of the sequence. Finally, $\circ_{\geq n}\varphi$ means that φ will not hold until the n$^{\text{th}}$ state of the sequence.[11] These are defined inductively:

$$\circ_{=n}\varphi = \begin{cases} \varphi & \text{if } n = 0 \\ \circ(\circ_{=(n-1)}\varphi) & \text{if } n > 0 \end{cases}$$

$$\circ_{\leq n}\varphi = \begin{cases} \varphi & \text{if } n = 0 \\ \circ_{\leq(n-1)}\varphi \wedge \circ_{=n}\varphi & \text{if } n > 0 \end{cases}$$

$$\circ_{\geq n}\varphi = \begin{cases} \varphi & \text{if } n = 0 \\ \circ_{\leq(n-1)}\neg\varphi \wedge \circ_{=n}\varphi & \text{if } n > 0 \end{cases}$$

The previous extensions induce natural ones for the binary past and future operators. For example $\varphi\mathcal{F}^{\Diamond=2}\psi$ or $\varphi\mathcal{P}^{\circ_2}\psi$ have the obvious meaning.

[11] This notation is somewhat misleading in that we are *not* stating that φ will hold for all the states after the n$^{\text{th}}$. We are just saying that φ does not hold for all the states strictly less than n, and furthermore, that it holds in state n.