Hyperdocuments as Automata: Trace-based Browsing Property Verification

P. David Stotts
Computer Science Dept., University of North Carolina
Chapel Hill, NC 27599-3175, USA

Richard Furuta
Computer Science Dept., University of Maryland
College Park, MD 20742, USA

J. Cyrano Ruiz
Reliability Engineering Dept., University of Maryland
College Park, MD 20742, USA

Abstract

In many hypertext systems, meaningfully traversing a document depends on capabilities, features, and navigational aids that are part of the browser implementation. For example, if a reader browses to a node that has no out links, then backing up, or "warping" to the table of contents can allow the browsing session to continue.

If hyperdocuments are to become interchangeable among hypertext systems, rather than being readable only on the systems from which they are authored, one obvious but complex approach is to try and standardize on (most likely, very many) browsing features and behaviors, forming some standard union of the capabilities of current major implementations. This approach molds (or perhaps restricts) future systems, since new browsing "features" must then be worked into such a standard. An alternate approach, used in this paper, is to de-emphasized browser features and emphasize inherent document structure with browsing semantics. An author should be able to create document structure so that the desired meaningful access patterns are inherently allowed by links rather than by browser capabilities.

We present a method of analyzing the browsing properties of a hypertext document by examining the links alone. This method is not specific to any particular hypertext system or document authoring format. With it, an author can be certain that a document will allow particular access patterns when read on any browser implementation that has a single navigation operation: direct link following. The method requires a mental shift in how a hyperdocument is conceived 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 then used to verify that these specifications are met by the behavior of the links-automaton. We illustrate the generality of our technique by applying it first to a Trellis document, and then to a Hyperties document.

Key words: hypertext, place/transition nets, Petri nets, browsing semantics, synchronization, security, temporal logic, verification, access control, versions, model checking.

1 Problem and approach

Halasz, among others, has noted the need for structural search and query mechanisms for increasing the utility of hypertext nets. He identifies two major subtasks [16]: "...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 dynamic properties a document has, that is, what sequences of links a reader may be allowed to follow during browsing, but

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

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 *intentional 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 [14], 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 a document's linked structure satisfies the required property specifications. The emphasis 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 access the document. This approach is a start on meaningful system-independent data for hypertext and hypermedia.

In this report, we first explain our idea of browsing properties, and present a temporal-logic-based specification language for expressing such properties. We introduce a checking method for this language based on existing automated methods for verification of concurrent programs. We illustrate this verification method first on Trellis documents, and then on a Hyperties document, to emphasize the system-independence of the idea.

2 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 [32, 33, 34].

Though we will first explain our general verification techniques in the context of Trellis, we emphasize that the approach is applicable to *any* hypertext system's documents. 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. This 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 a finite state machine's state transition diagram. We will refer to this automaton as the hyperdocument's *links-automaton*.

This conceptual framework is illustrated in Figure 1. 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-only structure itself. The core functionality of a Trellis system is limited to a strict implementation of the transition rule for that automaton. We have obtained more power, more expressibility, by using a more general class of automaton, the place/transition net (Petri net),[1] rather than a finite state machine as the basic document structure. A place/transition net inherently allows parallel threads of activity (obtained with multi-head/multi-tail links), and also is not limited to expressing a finite number of states.

To illustrate more clearly the meaning of a links-automaton, consider the abstract document shown in Figure 1. In this structure, 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 Turing machine (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 of a document may wish to ensure that there is some path from each node back to the start node, for instance.

[1] We assume some familiarity with general net theory. Interested readers can get details in previous Trellis papers [32], or in summary texts by Murata [24], Reisig [29], and Peterson [26].

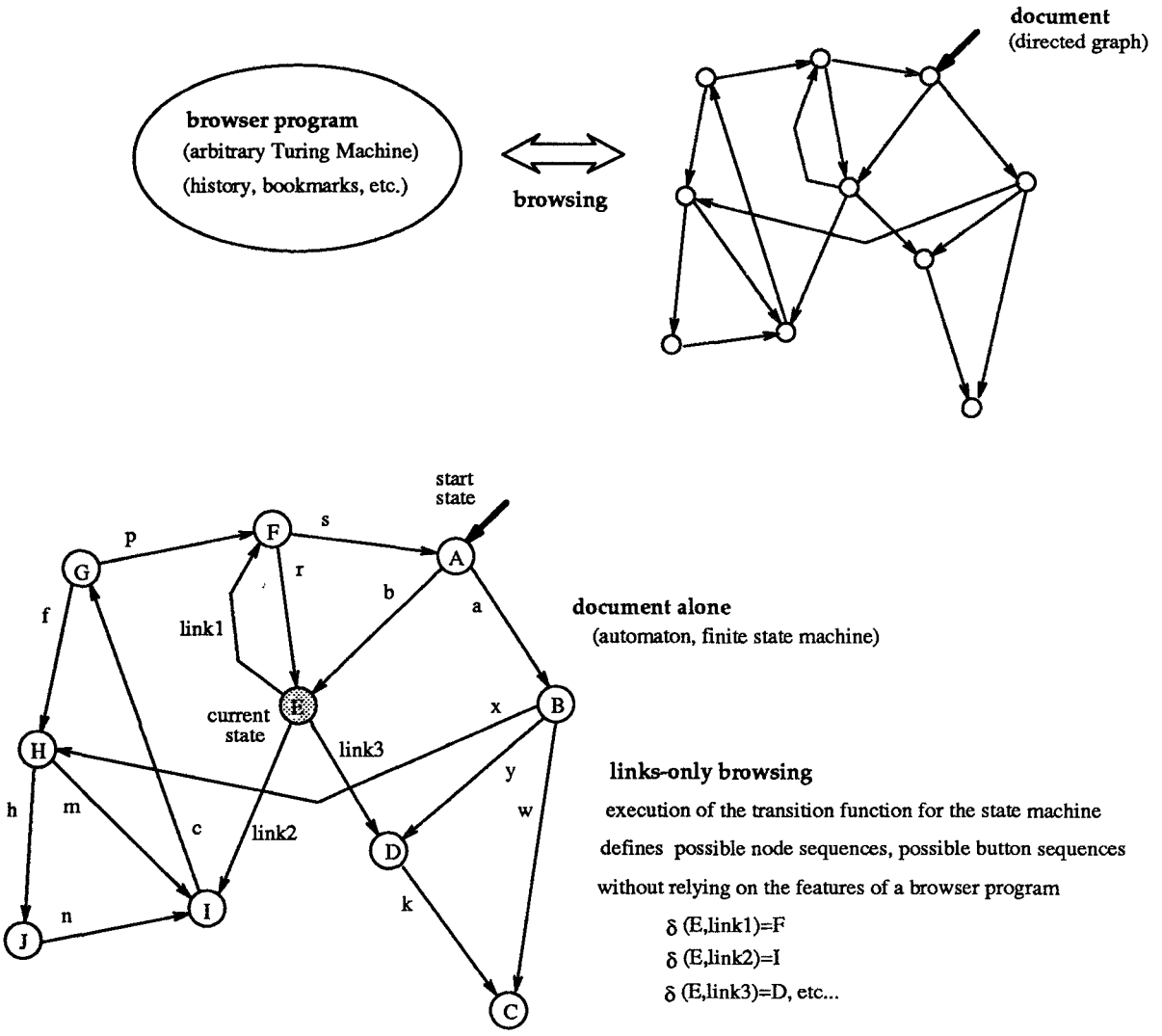

Figure 1: Traditional view of hypertext document, and automaton view.

3 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 each 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

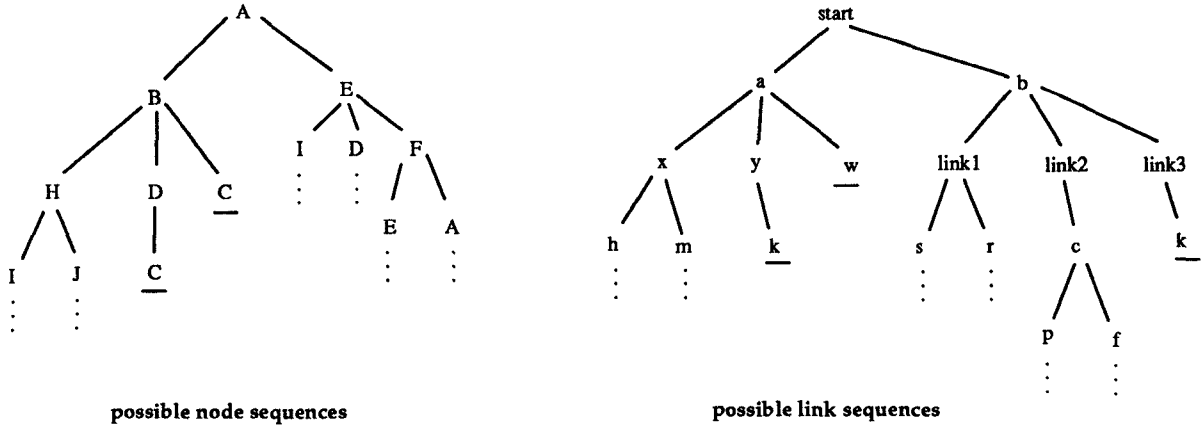

Figure 2: Branching event trees for links-automaton in Figure 1

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*.

In a branching temporal logic, formulae are interpreted as assertions about tree-like event spaces in which each event may have numerous distinct possible next events. Operators in such a logic allow statements about the structure of an event sequence, and also allow quantification over collections of possible event sequences in the tree. Branching temporal logic, then, is a natural way to express properties that hold for the browsing traces allowed by a hyperdocument.

3.1 Background on 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, 28], in artificial intelligence and cognitive science [18, 17, 20], in specification and verification of concurrent computations [2, 21] and in verification of network protocols and sequential circuits [23]. A complete survey of classical works in temporal logic as well as its current applications is found in [1].

In a branching temporal logic one introduces special logic symbols which allow formulation of assertions involving relative ordering as well as quantification over paths in the tree-like model of time. Various authors have proposed distinct syntaxes and semantics for branching time logics [19, 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 others.

Very little other work has been done applying temporal logic to hypertext problems. In one project, temporal logic has been used for structural queries of hypertexts based on directed graphs [3]. In this work, a formula describes a static property desired (as opposed to the trace-based, dynamic properties studied in our work), and a satisfaction algorithm is applied to locate a subgraph of the hypertext links that matches the formula.

4 HTL* and HTL

We give a brief introduction to the syntax and semantics of the specification language we have developed for expressing hypertext browsing properties. The notation is based on the previously mentioned temporal logic CTL*, and the efficiently verifiable subset CTL. We produce a new notation called HTL* and a corresponding subset called HTL, by adding operators that help express properties we feel are natural in a browsing context. Since CTL* is a complete notation, we deal with HTL* and HTL formulae by translating into appropriate CTL* formulae. HTL*will be introduced here primarily by example; a full syntax and semantics definition, and a description of the translation to CTL*, is reserved for a more detailed report.

CTL* has the well-known temporal operators $\Box$, $\Diamond$, $\circ$, U meaning respectively "always", "eventually", "next time", and "until". There are also the path quantifiers "forall" and "exists". HTL*adds direction to the CTL* notion of path quantifiers (like POTL) by placing an arrow above the respective quantifier

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

symbols. Thus in HTL* one may express "for all forward paths" ($\vec{\forall}$) or "there exists a backward path" ($\overleftarrow{\exists}$) for example. Such operators help in expressing properties like "when node X is seen, node Y must have been seen no more than 5 links previously;" the "links previously" part expresses a backward path.

Combinations of forward and backward path operators that commonly occur are abbreviated as $\mathcal{P}^{\Diamond}$ ("at some time in the past"), $\mathcal{F}^{\Box}$ ("at all times in the future"), $\mathcal{F}^{\circ}$ ("immediately after"), etc.

4.1 Example hypertext queries

We give a list of formulae in the context of hypertext. In all of these formulae we assume that we are referring to the initial state of the hypertext. Strings like "c.warning" and "b.option" are atomic predicates in the HTL* notation, and form part of the annotation of the model with basic information about what is "going on" in the hyperdocument at specific points of execution. For example, "b.option" asserts that a button named "option" is selectable in the current state; "c.warning" asserts similarly that a content element called "warning" is displayed in the current state.

- $\vec{\forall}(\Diamond$c.menu $\wedge$ c.menu $\mathcal{P}^{\Diamond}$ c.identification) is interpreted as "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."
- $\vec{\exists}\Diamond$c.warning which means "there is a browsing session encountering *warning*."
- $\vec{\exists}\Diamond$(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."
- $\vec{\forall}$(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".
- $\vec{\forall}\Box$(c.error $\leftrightarrow$ c.description) reads "for all browsing sessions, it is always the case that content element *error* is visible if and only if *description* is visible."
- $\vec{\forall}(\Diamond\varphi \vee \Box\neg\varphi)$ is a trivial tautology which states that in all browsing sessions either φ happens at least once or it never happens at all.
- $\overleftarrow{\forall}\Diamond_{=n}\varphi$ 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."
- $\vec{\forall}\Box\vec{\exists}\Diamond$c.menu means "wherever you happen to be, it is possible to eventually get back to the menu."
- $\vec{\exists}\Box\neg$b.confidential means "there is a way to browse the hypertext such that the button 'confidential' is never selectable".
- $\vec{\exists}\Diamond\vec{\forall}\Box(\neg$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. This is similar to level 3 of liveness for a transition defined in [25].
- $\vec{\exists}\Diamond\vec{\forall}\Box\neg(b.B_1, \ldots, b.B_k)$ where $B_1, \ldots, B_k$ are all the buttons, means that it is possible to fall into a 'deadlock.'
- $\vec{\forall}\Box\vec{\exists}\circ$c.end means "Wherever you are, you can immediately leave the hypertext" (assuming that this is the interpretation of content 'end').
- $\vec{\forall}$c.hello means "At the beginning of every browsing session, you must encounter the 'hello' content element."

5 Verifying the validity of HTL* Formulae

We now discuss how to determine if a hypertext document has browsing properties that have been specified as HTL* formulae. Broadly, we accomplish this by applying an algorithm developed by Clarke for verifying CTL formulae on finite state machines. There are several steps in the approach. We first must express properties in the language HTL, a restriction of HTL*; we then must translate HTL formulae into equivalent CTL formulae. We must also determine how to produce an appropriate finite state machine from a hyperdocument. After this, Clarke's model checker can be successfully employed for browsing property verification.

Since the HTL is translated into CTL for verification, we give subsequent results in terms of CTL only to simplify the presentation.

5.1 The model checking approach

In model checking, a state machine (the model) is annotated with atomic properties that hold at each state in the model (such as "content is visible" or "button is selectable"), and then search algorithms are applied

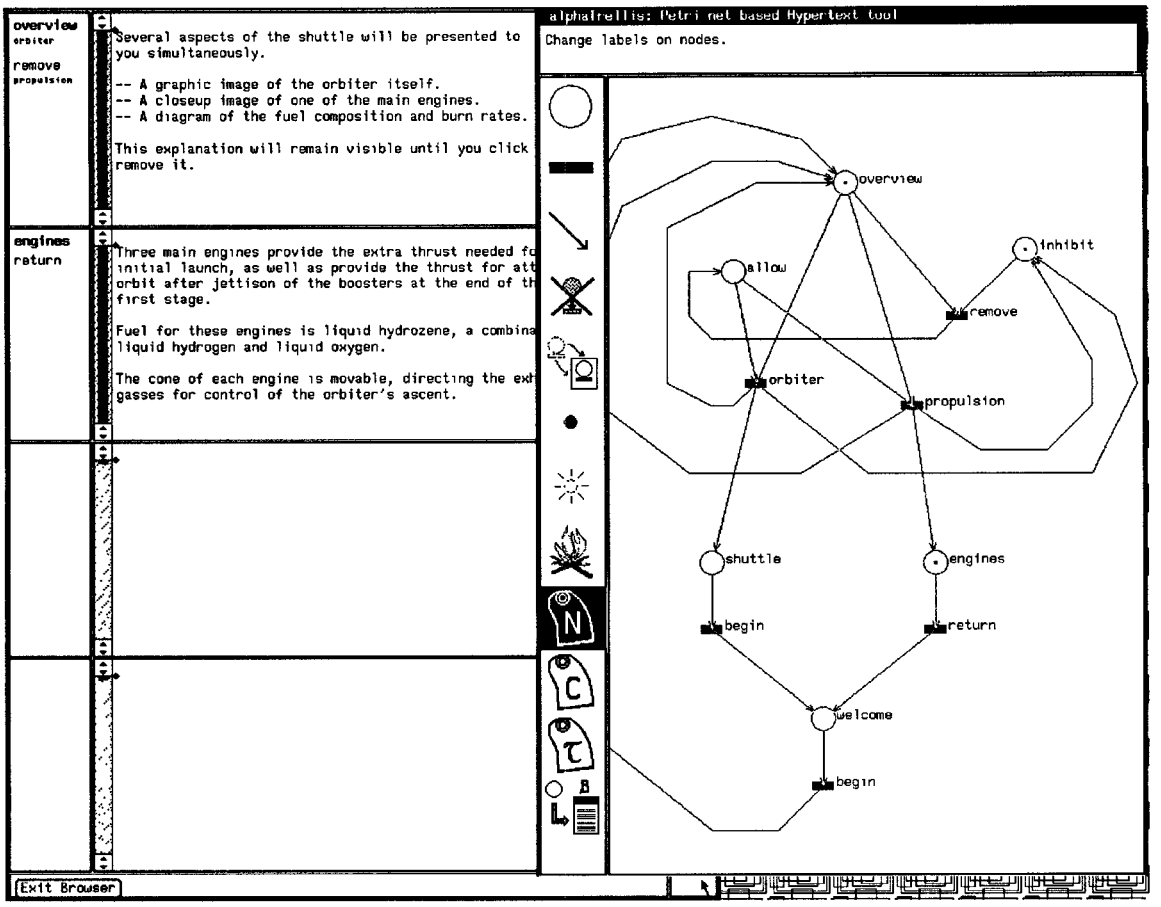

Figure 3: Small αTrellis document.

to see if the subformulae of a formulae hold at each of the states. By composing the truth values of the subformulae, one may obtain a truth value for the entire formula.

For Trellis, we obtain a useful state machine from the *coverability graph* explained in an earlier Trellis paper [32]. (We should note that generating this state machine can require time and space that is exponential in the size of the Petri net.) For other hypertext systems, the links-automaton directly is an appropriate finite state machine.

In the general case (and especially with very long formulae) checking of CTL* formulae, while possible, may have exponential complexity. The restricted language CTL allows a very efficient checking algorithm to be developed, however. Clarke *et al.* have extensively studied the model checking problem for CTL [10, 11, 5].

The simplifying characteristic of CTL is that certain of the temporal operators must always be paired, reducing the number of combinations that must be searched in the model. For example, each path quantifier must be immediately followed by exactly one of the operators $\circ$, $\mathcal{U}$, $\Box$, or $\Diamond$. Since this pairing is not required in CTL*, the possible CTL formulae form a subset of the possible CTL* formulae. Thus there are some browsing properties that cannot be efficiently checked with this approach. In general, however, CTL is sufficient to express a wide variety of useful properties.

5.2 Examples of property verification

In this section we demonstrate how we have applied a model checker for CTL developed at Carnegie Mellon by Clarke to verification of browsing properties. We first illustrate the method for Trellis, and then demonstrate its generality by analyzing a well-known Hyperties document. We should note that though the examples were all direct output of the model checking software, we have not represented the notation directly, as the model checker requires an ascii representation for the temporal logic symbols.

Sample Trellis document and queries

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.

In our terms, the Petri net is the links-automaton of this hyperdocument. It is not, however, in a form that is normally thought of as a finite state machine. In order to obtain a finite state machine for model checking, we compute the coverability graph of the net and use that. The Petri net shown gives rise to a coverability graph of 11 states. From this, we generate an 8-node finite state machine, the links-automaton for this document.[3] This model can then be queried for the desired browsing properties with the model checker. We illustrate several such queries here.

- 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)
  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)
  The formula is FALSE.
- Is there some browsing path that reaches a point at which no buttons are selectable?
  $\vec{\exists}\Diamond$($\neg$B_begin $\wedge$ $\neg$B_remove $\wedge$ $\neg$B_propulsion $\wedge$ $\neg$B_orbiter $\wedge$ $\neg$B_begin2 $\wedge$ $\neg$B_return)
  The formula is FALSE.
- Is there some browsing path that will eventually simultaneously show the "overview" panel and the "engines" panel?
  $\vec{\exists}\Diamond$(C_overview $\wedge$ 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)
  The formula is FALSE.
- Is it possible during browsing to see both the "welcome" and "engines" panels on the same screen?
  $\vec{\exists}\Diamond$(C_welcome $\wedge$ 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?
  $\vec{\exists}\Diamond$(C_inhibit $\wedge$ C_allow)
  The formula is FALSE.

[3]The state count drops because construction of the coverability graph produces a structure with repeated nodes in it. These nodes are removed when converting to the FSM input format for the model checker.

- 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]))
  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.

A larger Trellis document

The model checker has also been tested on the larger Trellis document shown in Figure 4. The state machine derived from this net contains nearly a thousand states. Using a DECstation 5000/25, the performance of the verification software on formulae like those discussed previously was mostly on the order of a few seconds, with the most complicated query we tried requiring about 25 seconds. We suspect that authors of interactive documents will find such timing not at all unreasonable for establishing the presence or absence of critical browsing properties, and we hope that future, less primitive systems, might exhibit improved performance as well.

A Hyperties document

To illustrate that these browsing verification concepts are not specific to Trellis-based systems, but are indeed general, we have applied the approach to a Hyperties [30, 22] document. Specifically, we have extracted the link structure from the Hyperties version of ACM's *Hypertext On Hypertext*, and use the resulting directed graph as the links-automaton and as the finite state machine for the model checker. The extraction of links from the Hyperties storage format, and the conversion of those links into the state machine input format required by the model checker, were both automated procedures. The model contains 298 states (content nodes), and over 700 links. Here are the results of some sample queries, each of which was answered by the model checker in less than two seconds on the DECstation 5000/25.

- 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?
  $\vec{\forall}\Box(\vec{\exists}\Diamond$(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

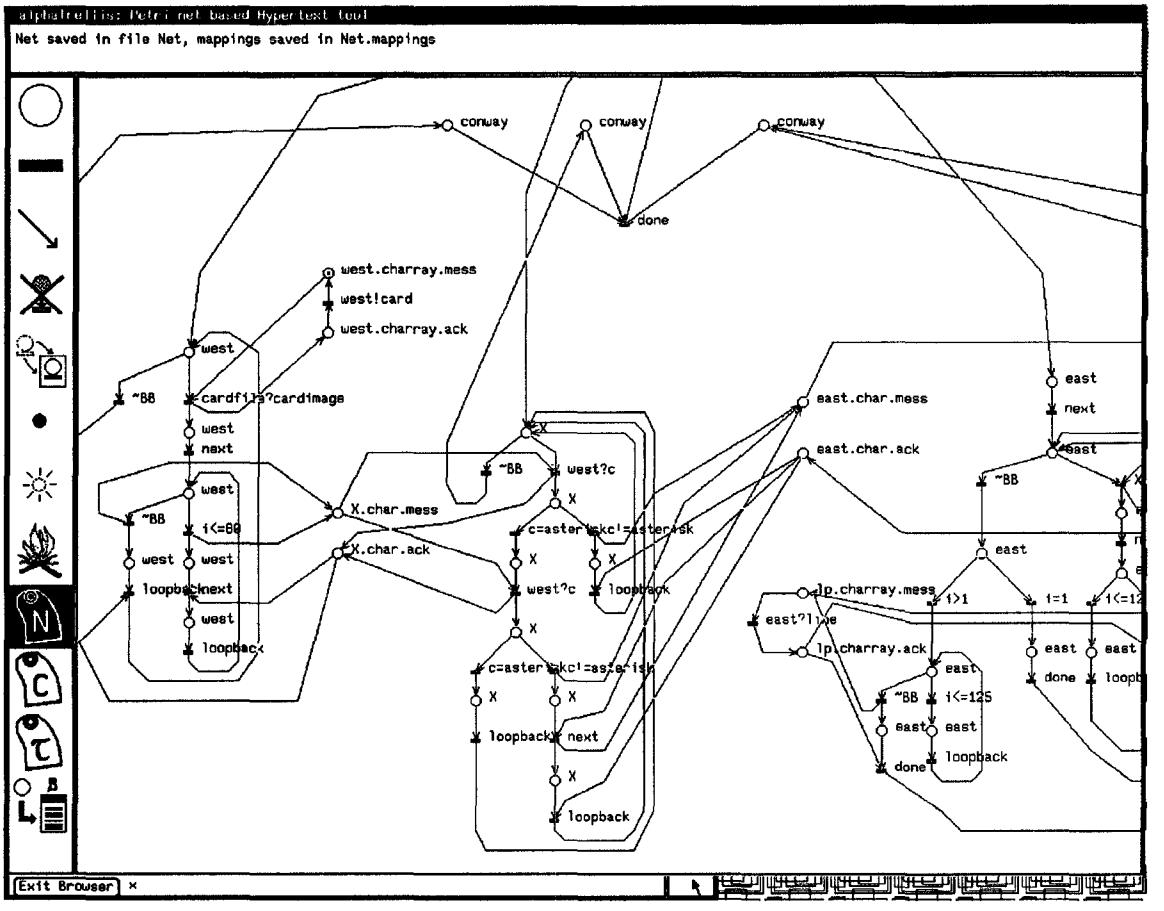

Figure 4: CSP program as browsable Trellis document

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?
  $\vec{\forall}\Diamond$(C_tableofc)
  The formula is FALSE.
  $\vec{\forall}\Diamond$(C_tableofc)
  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?
  $\vec{\exists}\Diamond$(C_introduc $\wedge$ $\vec{\exists}\circ(\vec{\exists}\Diamond$(C_introduc)))
  The formula is FALSE.
  Since the *Introduction* is the first node on all paths, the failure of this property implies that 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?
  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?
  $\vec{\forall}\Box(\vec{\exists}\circ$(C_tableofc))
  The formula is FALSE.
  $\vec{\exists}\Box(\vec{\exists}\circ$(C_tableofc))
  The formula is TRUE.
  This first property might well be one an author wants a document to have, though it is strong. Interestingly enough, though the "one hop away" property for *TOC* does not hold for all paths, it

does hold for at least one path in this document. The falsity of the first property does not necessarily imply the truth of the second.

- On all paths from the initial node, is it always the case (at all nodes) that when the *Bibliography* is displayed some path eventually leads from it to the *TOC*?
  $\vec{\forall}\Box(\neg$C_bibliogr$\vee\vec{\exists}\Diamond$(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.

6 Conclusions

In summary, we have presented a method by which an author may specify properties that should exist in the sequences of events that occur during browsing, and a method by which an author may verify that a hyperdocument does, or does not, have these properties. We have explained the links-automaton view of a hyperdocument that makes our techniques possible, and showed that this concept is a general one, making the techniques applicable to all hypertext systems. We also showed how to apply the technique to links-automata for documents in two existing systems, Hyperties and Trellis.

Clearly, practical application of the verification techniques reported here requires an interface that allows an author to express properties without having to be an expert in temporal logic syntax and semantics. We have not yet produced such an application, and we view the design of such an interface as an important research problem. Another interesting research problem is how to use HTL* formulae *a priori*, as a guide during document development rather than simply as an *a posteriori* check after authoring.

References

[1] Galton Antony, editor. *Temporal Logics and their applications*. Academic Press, Harcourt Brace Jovanovich, Publishers, 1987.

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

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

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

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

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

[7] Brad Campbell and Joseph M. Goodman. HAM: A general purpose hypertext abstract machine. *Communications of the ACM*, 31(7):856–861, July 1988.

[8] S. Christodoulakis, F. Ho, and M. Theodoridou. The multimedia object presentation manager of MINOS: A symmetric approach. In *Proceedings of ACM SIGMOD '86*, pages 295–310, May 1986. Washington, DC.

[9] S. Christodoulakis, M. Theodoridou, F. Ho, and M. Papa. Multimedia document presentation, information extraction, and document formation in MINOS: A model and a system. *ACM Transactions on Office Information Systems*, 4(4):345–383, October 1986.

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

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

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

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

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

stitute for Advanced Computer Studies, January 1991.

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

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

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

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

[19] L. Lamport. "sometime" is sometimes "not never": on the temporal logic of programs. In *In Proc. seventh ACM Symp. on Programming Languages*, pages 174–185, 1980.

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

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

[22] Gary Marchionini and Ben Shneiderman. Finding facts vs. browsing knowledge in hypertext systems. *Computer*, 21(1):70–80, January 1988.

[23] B. Moszkowski. *Reasoning about digital circuits*. Ph.D. dissertation, Standford University, CA, 1983.

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

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

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

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

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

[29] Wolfgang Reisig. *Petri Nets: An Introduction*. Springer-Verlag, 1985.

[30] Ben Shneiderman. User interface design for the Hyperties electronic encyclopedia. In *Proceedings of Hypertext '87*, pages 189–194, November 1987. Published by the Association for Computing Machinery, 1989.

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

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

[33] P. David Stotts and Richard Furuta. Temporal hyperprogramming. *Journal of Visual Languages and Computing*, 1(3):237–253, 1990.

[34] P. David Stotts and Richard Furuta. Browsing parallel process networks. *Journal of Parallel and Distributed Computing*, 9:224–235, 1990.

[35] I. Suzuki and H. Lu. Temporal petri nets and their application to modeling and analysis of a handshake daisy chain arbiter. *IEEE Trans. Comput.*, 38(5):696–704, 1989.

[36] Z. Wolper. Temporal logic can be more expressive. In *In Proc. of the twentysecond IEEE Symp. on Foundations of Computer Science*, pages 340–348, 1981.

[37] Polle T. Zellweger. Directed paths through collections of multi-media documents. In *Hypertext '87*, November 1987. Position paper.

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