Structure Analysis for Hypertext with Conditional Linkage

Jean-Hugues Réty

CWI, P.O. Box 94079,
1090 GB Amsterdam, The Netherlands
E-mail: jean-hugues.rety@cwi.nl

ABSTRACT

We propose a structure analysis and proof framework for hypertext with conditional linkage. This framework can provide hypertext systems with a powerful and simple tool to help the writer to maintain control over the browsing semantics of her hypertext. We briefly present the outline of a possible implementation in PROLOG.

KEYWORDS: Conditional link, structure analysis

INTRODUCTION

Conditional linkage appears to be a useful extension of the notion of link, especially for literary writing (see [2]) where the writer may for instance want some links to depend on the history of previously visited nodes, or for adaptive hypertext where links may be hidden or annotated depending on the identity of the reader, the history, or some other data. In other words, the writer may want to impose constraints on the browsing behavior of the reader. Conditional linkage is a simple way to do so. A major issue is then to provide the hypertext writer with efficient, high level control over the structure of her writing. The well-known "lost in hyperspace" notion is not to be restricted to the reader: it may be a very difficult problem for the writer to keep a global view over the complexity of links. Most hypertext systems include some graphic tools permitting the visualization of (some part of) the structure of the hypertext but these tools are user-oriented and not precise enough. Moreover, they can hardly accommodate conditional links. In order to help the writer maintain control over the link structure, we propose an automated structure analysis and proof framework for hypertext with conditional linkage.

HYPERTEXT STRUCTURE

Existing models of hypertext such as the Dexter Hypertext Model are too general and too complex for our purpose: we need to focus on the structure independently from other features, to define what is called in [3] a *links-automaton*. We thus restrict ourselves to a simpler definition of hypertext intended to serve as a basis of our work. Conditional linkage is introduced with a notion of *state*: a link can be followed only if some condition on the state is satisfied. A state is some data that results from the browsing of the hypertext by the reader. For sake of brevity, we will restrict in this paper states to the historical list of previously visited nodes (the history, for short). Notice that, in general, states may have a more complex structure (for instance, some information introduced by the reader when reading some other nodes and answering questions). Browsing a hypertext starts with an initial state (here, the empty history). Then, each time the reader follows a link, the state is updated (here, the current node is added to the history).

In what follows, a *link* is a triple *(n,m,cond)* where *n* and *m* are nodes and *cond* is a condition (first-order logic predicate) on states. Intuitively, the link *(n,m,cond)* means that, with current state *S*, the reader can move from node *n* to node *m* if *cond(S)* holds. Since we are only interested in the hypertext structure, we abstract from any notion of content of a node or display information. Thus, a node is here simply characterized by a (unique) identifier.

A *hypertext* is formed of a set of nodes and a set of links. It may contain a particularized set of *entry nodes* from one of which the reader must start her reading. If not, all nodes are considered to be entry nodes.

BROWSING ANALYSIS

The writer wants to impose constraints on the browsing behavior of the reader. Conditional linkage appears as a simple and powerful tool in this task, and indeed, is used by hypertext writing systems like Storyspace or adaptive hypertexts systems like, e.g., Interbook. But as the number of nodes and the complexity of links increase, it becomes more and more difficult for the writer to maintain a global control over the different readings her writing allows for. Constraints (conditional links) imposed by the writer take place at the level of links, at the "microscopic" level, but the writer wants her writing to satisfy some "macroscopic" structural properties like e.g.: "node *n* reporting the hero's marriage must be read before node *m* reporting his death". Such properties may be very difficult to check, due to the complexity of the structure. If the only way to access node *m*

Hypertext 99 Darmstadt Germany

is a link from *n* to *m*, this is simple. But what about if ten links are pointing to node *m*, and hundreds of different paths exist for the reader to go from node *n* to node *m*? Then, what we want to check is: does every reading visiting node *m* meet *n* before reaching *m*? It may be the case that thousands of such readings exist… and clearly, the writer needs some help. We propose below an automated framework for verification of properties of the structure of (conditional linkage-based) hypertexts. It is based on a PROLOG engine.

Let us consider a hypertext *h*. We define a PROLOG predicate **reading$_h$(N,M,H)** where **N**,**M** are nodes, and **H** is a list of nodes (the history). Its intuitive meaning is: **reading$_h$(n,m,[node$_k$,..,node$_1$])** holds if **node$_1$=n**, **node$_k$=m**, and the sequence of nodes **node$_1$,…,node$_k$** is a valid reading of the hypertext *h*, i.e., a reader can browse the hypertext from node **n** to node **m** through the sequence of nodes in **H**. The predicate **reading$_h$** is defined in PROLOG by the following clause (in bold font):

```
reading_h(F,M,[M|H]) :-  % [M|H] is a reading from
                         % F to M if:
   link(N,M,Cond),       % there is a link (N,M,Cond), and
   reading_h(F,N,H),     % H is a reading from F to N, and
   check(Cond,H).        % H satisfies condition Cond.
```

and by the fact: **reading$_h$(N,N,[N]).** Then, for each link *(n,m,cond)* of *h*, we have the fact: **link(n,m,cond).** Finally, most conditions on the history are simply described with standard PROLOG list predicates. Consider for instance the link *(n1,n2, n6 has not already been visited)*. Then, this condition is checked with: **not(member(n6,H)).** Notice also that, since the hypertext structure may contain cycles, the evaluation of the predicate **reading$_h$** may not terminate. Adding a test on the history solves this problem. We omit technical details here.

This framework allows the writer to check for structural or browsing properties of the hypertext: every query expressible as a logic programming query can be ran. We give below some examples of such queries with their translation into first order logic and logic programming queries (in bold font). Let *n,m,p* be nodes.

- Is *m* accessible from *n*? $\exists H reading_h(n,m,H)$
  reading$_h$(n,m,H).
- Does there exist a reading from *n* to *m* going through *p*?
  $\exists H(reading_h(n,m,H) \wedge member(p,H))$
  reading$_h$(n,m,H),member(p,H).
- Does every reading starting at *n* visit *p* before *m*?
  $\forall H(reading_h(n,m,H) \Rightarrow member(p,H))$
  not(reading$_h$(n,m,H), not(member(p,H))).
- What is the shortest reading *H* from *n* to *m*?
  $reading_h(n,m,H) \wedge \forall L(reading_h(n,m,L) \Rightarrow H \leq L)$
  findall(M,reading$_h$(n,m,M),L),minpath(L,H).

member and **findall** are standard predicates of PROLOG systems. **findall(M,reading$_h$(n,m,M),L)** returns the list **L** of all the readings from *n* to *m*. **minpath** is simply defined using standard list manipulation predicates.

This framework can result in an automated partial correctness proof framework. Suppose that the writer gives a set of properties she wants her hypertext to satisfy. This set of properties forms a partial specification of the structure of the hypertext. Then, the PROLOG engine will automatically check the conformity of the actual hypertext with respect to that specification. We think that authoring systems using conditional links could simply define such a tool by embedding a PROLOG engine.

Another issue concerns diagnosis. Whenever the hypertext does not satisfy the properties given by the writer (e.g., if the PROLOG engine answers that there exists a reading where the hero dies before he gets married), then the question is: What's going wrong? What link(s) is responsible for this inconsistency? We think that hypertext could here take benefit from the extensive work concerning error diagnosis in logic programming.

RELATED WORK

[3] proposes a verification framework for hypertext whose aim is close to our but where browsing properties are expressed in a modal logic. On the other hand, some hypertext querying frameworks were designed for information retrieval. In particular, [1] proposes an enriched modal logic based querying language. In contrast, we argue that first order logic is the right tool for our concern. It benefits from its simplicity and from the well-understood computing mechanisms of PROLOG.

CONCLUSION

Links, and more specifically conditional links, have a strong logical flavor. We propose an automated proof framework for browsing properties of hypertext with conditional links. This approach is attractive because of its genericity and its simplicity. First, every query expressible as a logic programming query can be computed; we think this tool to be general enough to cover most questions the writer may want to get an answer concerning the structure of her writing. Second, some simplified interface languages may be designed for those not confident with logic programming and first order logic in general. In this paper, states are restricted to histories but our framework can be extended to other, more complex notions of state.

ACKNOWLEDGMENTS

We are indebted to A. Eliens, L. Hardman and N. Guimarães for some helpful comments.

REFERENCES

1. Beeri, C. and Kornatzky, Y. A Logical Query Language for Hypertext Systems. Proc. of ECHT'90, pp. 67-80.

2. Kendall, R. Hypertextual Dynamics in *A Life Set for Two*. Proc. of Hypertext'96, pp. 74-83.

3. Stotts, P.D., Furuta, R. and Cabarrus C.R. Hyperdocuments as Automata: Verification of Trace-Based Browsing Properties by Model Checking. ACM Trans. on Information Systems, 16(1):1-30, 1998.