DYNAMIC HYPERTEXT SYNTHESIS FOR INFORMATION RETRIEVAL

David Bodoff
Stern School of Business
New York University

Working Paper Series
STERN IS-93-32

Dynamic Hypertext Synthesis for Information Retrieval

David Bodoff
Information Systems Department
Leonard N. Stern School of Business
New York, N.Y. 10012

Abstract

Hypertext navigation alone is insufficient for efficient Information Retrieval (IR). Previous attempts to combine IR techniques with hypertext have been confined to the pre-authored structure of a document. In this paper we extend computer-science methods to synthesize a tailor-made hypertext document in response to each user's query. The synthesis technique can also be used to automatically create a pre-authored hypertext document according to an author's specifications.

Introduction

As the amount of available electronic information continues to grow, the problem of information retrieval grows in urgency. Two paradigms for IR are database queries and hypertext browsing. Information which is easily stored in database form is accessible to users looking for isolated pieces of information. On the other extreme, users who are not sure what they're looking for can browse through a hypertext [15].

But many cases fall in between, where the user knows what he's looking for, but he's not looking for an isolated piece of database-like information. Rather, real users frequently seek a more thorough **understanding** of an unfamiliar concept. For example, an executive interested in adopting matrix management in his organization needs to understand the concept, its goals and risks, its implementation, etc.

This type of information requirement has three defining features which make standard IR techniques inadequate. First, the user's need for understanding requires that he "read-up" on his topic; this involves *numerous* elements of *textual* description. Second, he has an *angle*. That is, he is interested in some -- but not all -- aspects of the topic. Our manager, for example, may be uninterested in the history of matrix management or in certain research questions. Third, his is not the only conceivable angle. Other users (e.g. historians and researchers) may be interested in different aspects of the topic.

Traditional Information Retrieval (IR) techniques are inadequate for this type of information requirement. First, we are limited to the inelegant methods of *text* retrieval. Traditional keyword-based methods are crude: Since a keyword is attached to a whole document, it is necessarily a gross averaging of the document's varied contents. Because of this inaccuracy, much relevant information is hard to find. On the other hand, whole documents are retrieved in response to a query. This imprecision results in presentation of much superfluous information [9]. These are not limitations of keyword techniques pre se. Rather, any IR technique which treats a document as an atomic unit will result in the problems of hard-to-find and irrelevant information. These problems are exacerbated by the differing *angles* which each user brings to a topic. No single document-labeling scheme is best at limiting superfluous and hard-to-find information for every possible *angle*. The information sought by each user is found in bits and pieces of many documents.

Suppose, on the other hand, that each article were decomposed into individual thoughts or nodes, and that each node were labeled with a keyword. Using this method, we will have controlled the problems of superfluous and hard-to-find information, since this technique is both more precise and more accurate. Moreover, freed from the constraint of treating a document as an indivisible unit, we could employ formal methods to decompose it into nodes such that typical queries yield a minimum of irrelevant and redundant information [19]. The problem now, however, is lack of scope: The manager must now go fishing in the database countless times, to piece together all the *numerous* bits of relevant information.

The ability of hypertext to combine individual nodes and groups of nodes ("states") into an organized structure seems promising. We can refer accurately and precisely to individual nodes and states, and we can achieve broad scope by collecting these into a larger hypertext structure. The problem is how to automatically retrieve for each user the nodes he wants, combine them appropriately into states, and organize these states into a hypertext structure.

In this paper, we present a method to accomplish this task. In response to each user's query, our method automatically retrieves all of the relevant nodes and organizes them into a hypertext structure. In our approach, the query itself specifies both the content and structural properties of the desired hypertext. An algorithm retrieves the relevant nodes and synthesizes a hypertext structure which exhibits those structural properties.

The remainder of this paper is organized as follows: First, we review related research, its contributions and shortcomings, and outline our proposed approach. In section two we introduce two very different uses for our method. Section three gives technical background necessary for understanding our approach. Section four provides more details of our algorithm, using a running example. The paper ends with a discussion of conclusions and future research.

Section One: Related Literature

Two research areas relate to the retrieval and organization of information nodes into a hypertext document. The first area addresses information retrieval in hypertext. The second area addresses automatic generation of hypertext. As we will see, each of these approaches taken alone is inadequate for the type of information requirement described above. Taken together as in our approach, they present a viable approach for meeting such information requirements.

1.0 Information Retrieval in Hypertext

There are two streams in this research[1]. The first stream takes a user's query and a pre-authored hypertext, and attempts to find a good starting point for browsing, given a user's query. These attempts may utilize traditional IR techniques to rank the similarity of each node's contents to the query [23]. Garzotta et al. [13] further provide hierarchical indexes for access into the hypertext. In addition, the pre-authored structure -- i.e. the meaning of links -- may be exploited to help rank a node's similarity to the query [10], [18]. Finally, the user's query may refer explicitly to the hypertext structure to search for structural patterns [2], [8].

These approaches do provide IR access into a hypertext. But they are all limited to the pre-authored document structure. A pre-authored document cannot possibly contain a web of nodes to correspond exactly to every conceivable query's *angle* on the information; the original author must insert a small number of "related-to" links, and his choices "may express an arbitrary and debatable judgment" ([14] p. 14). Unless a user's point of view exactly corresponds to the author's, the problems of superfluous and hard-to-find information recur.[2]

The second stream of research, rather than providing access into a hypertext, retrieves individual nodes [11], [19], or organizes them into a new linear structure known as a guided tour [15]. A guided tour may be a *good* linear sequence, and this may suffice for some uses. But even a good linear sequence does not provide the browsing capabilities with which the user can incrementally refine his search.

In summary, these two approaches do begin to combine IR with hypertext. However, these approaches either provide the original pre-authored branching structure (first approach), or provide no branching structure at all (second approach). This is understandable, because of the difficulty of dynamically creating a new hypertext structure for the dynamically retrieved nodes. We turn now to review the literature of dynamic hypertext generation.

1.1 Automatic Generation of Hypertext

Assuming we had retrieved all the individual nodes relevant to this user, we would need a method to automatically structure them into a meaningful hypertext document. Most approaches to automatic link-generation take advantage of the underlying structure of the domain [23], [4]. Bieber's bridge laws, for example, explicitly "describe the internal structure of the information system" ([4], p. 393), and the generated hypertext reflects that structure. This works well to relate the inputs and outputs of information **in a system**. But whereas an information *system* has a clear structure, arbitrary text does not. Thus, others [17], [9] suggest using syntactic hints in the text of nodes to generate links. A significant limitation of this approach is its complete insensitivity to the individual user's point of view. Even a perfect

[1] Reviewers of earlier versions of this paper were uncomfortable with this characterization of cited works into these two 'camps'. The research contributions of these works extend far beyond this characterization. But I believe the categories are nonetheless useful to map out the various possible meanings of 'combining' IR with hypertext.

[2]HDM2 [15] supports a filtering mechanism to alleviate the problem of superfluous information. The problem of hard-to-find information remains.

algorithm of this kind can only be as good as manually created pre-set links; it cannot dynamically link nodes from the point of view of each query.

1.2 Information Retrieval and Link Generation Combined

We propose combining the methods of *IR from hyperbases* with automatic link generation to provide a more adequate response to each user's information needs. Formal techniques guide our decomposition of text into nodes [19]. The best available IR techniques retrieve the individual relevant nodes for each user's query. Moreover, the meaning of pre-authored links can be utilized to help identify relevant nodes, as in [10]. However, once the individual relevant nodes are identified and retrieved, the pre-authored structure is discarded. (In fact, the retrieved nodes may not have been connected at all in the pre-authored structure.) Automatic link generation techniques are then applied to the retrieved nodes, creating a navigable hypertext document for each query. This is the approach we adopt in this paper, and we expect to see more research into this useful combination of methods.

But as we have noted, current techniques for automatic link generation remain inadequate. In particular, they remain insensitive to the particular *angle* of each user query. Thus, automatically-generated links can, at best, be only as appropriate as manually pre-authored links. What is needed is a method to automatically create links which make sense for the particular query posed.

Our new approach to link generation is to allow (require) the user to include in his query structural specifications to guide construction of a browsable hypertext from the retrieved nodes. Rather then retaining the pre-authored hypertext structure, each query defines new hypertext links for the requested content. The advantage of this particular approach is that, rather than hoping to automatically guess the user's angle on the topic and a correspondingly appropriate linked structure, the user himself requests the structure he wants. The automatically generated structure is always right.

It should be clear that our contribution is not in the area of IR indexing. Any method of IR ultimately depends upon the original decomposition of information into coherent units, and appropriate indexing of those limits. See [19], for example, for a discussion of the decomposition problem: "The question is: How can we design the information groupings so commonly selected predicates will encounter a "reasonable" amount of redundancy or irrelevance?" (ibid., p.5). In this paper, we assume that a reasonable decomposition has been achieved. Furthermore, we assume some sort of reasonable keyword-based indexes. Our contribution lies not in refining these fundamental IR techniques, but in combining them with computer science techniques so that the result of a query is a coherent, tailor-made hypertext.

Section Two
Two Roles for Automatic Synthesis

The method we propose serves two very different functions. Viewing the user's input as a *query*, hypertext synthesis is a method of IR. Viewing his input as a hypertext *specification*, our synthesis method helps automate the authoring of hypertext documents.

Automatic Synthesis as Authoring Tool

While we view the real importance of hypertext as a tool for general information retrieval, many applications do lend themselves to pre-authored documents. These documents are being developed for use within organizations and for commercial use, and methods and tools for good document design are being researched and developed [1], [12], [16], [9].

Authoring a large hypertext document is a complex process. Suppose, for example, that we wish to compose a hypertext to teach a student about databases. The author would want his document to possess certain properties, such as:
From a display of topic headings, there is a path which eventually leads to each topic in the headings list;
For each topic, an introduction must precede any substantive material;
From any advanced topic, there is a path leading to an example;
From an advanced example, there is at least one path to an intermediate example;
etc.

We call these desired properties '*specifications*'. It is easy to see that even a small number of specifications becomes unmanageable without automated support. As we see below, Stotts et al. [21] suggested an authoring tool which would ascertain -- after the fact -- whether a document satisfied the specifications. Aside from this suggestion, we know of no authoring tools to support this complex aspect of the authoring process.

The method we propose in this paper completely automates this part of the authoring process. Given a set of specifications as in the above example, we construct a graph to show how the information must be structured. In a second stage, the algorithm retrieves actual information nodes from a hyperbase and organizes them into such a structure.

Automatic Synthesis as Information Retrieval

But we emphasize the IR use of our method because we agree with the prevailing consensus that navigation alone is not a sufficient answer to the general IR problem. Halasz [16] spoke of dynamic virtual nodes and links which are intentionally-defined by "specifying a description of their components". These virtual nodes and links would be powerful even in a pre-authored virtual document, as, for example, the intentionally-defined links would be born and die with the evolving hyperbase of nodes. But as Halasz points out, queries combine with virtual components as a powerful IR technique. Queries can define the desired nodes and links in terms of their contents, connections, etc., and the result of a query would be a dynamically-formed structure among dynamically-retrieved nodes. This is the concept we have adopted and realized as a powerful method for IR.

In this view of hypertext synthesis, the user's input is a *query*. Suppose, for example, that a computer science professor were doing research in databases. Rather than providing all database students and researchers with one hypertext on the subject, we suggest providing a hyperbase of unconnected information nodes, plus a query language. The professor's query might look something like this:

I want information on object-oriented databases;
I want that all nodes which discuss a query language be immediately followed by an example;
that from any informal treatment of a data model there must exists a path to the corresponding query language;
etc.

In this way, a user describes the content of interest and certain structural properties of the hypertext document he would like to browse. Our method provides a tailor-made hypertext in response to each user query. This allows a user to browse through a hypertext which contains all and only the information he needs in an appropriate navigable structure.

Section Three Technical Background

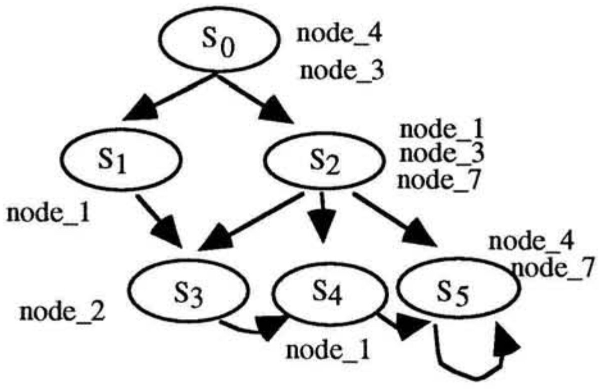
Our work is based on two sources: Stotts et al. [21] originally suggested (for future research) using temporal logic formula to help automate hypertext synthesis. Ben-Ari et al. [3] provide one of the simpler algorithms for automatic synthesis of computer 'synchronization skeletons' from temporal logic specifications. Combining these ideas, we have extended Ben-Ari's algorithm to apply to automatic synthesis of hypertext. Each of these sources is discussed below.

3.1 Hypertext Defined

We view a hypertext document as a graph of states. In each state, a small number of nodes and links is visible. From each state is accessible a number of possible successor states. Let
N denote a (finite) set of nodes
S denote a (finite) set of states
S_0 denote a special start state
T: $S \rightarrow 2^S$ denote a complete accessibility relation, i.e. from each state is accessible a nonempty set of states
M: $S \rightarrow 2^N$ denote a mapping from each state to a set of nodes visible at that state

Then the structure $H = < S_0, T, M >$ denotes the *reachability graph* of a hypertext which starts in S_0 and which can then enter states $S_0..S_{n-1}$ according to T.

A reachability graph can be represented pictorially in this way:



3.2 Properties of Hypertext

The reachability graph of a hypertext document can be viewed dynamically as a finite state machine representing the possible execution paths of a browsing session. Realizing this dynamic interpretation of the reachability graph, Stotts et al. [21] proposed using the graph to answer true-false queries about the possible executions of the hypertext, on the pattern of [7].

For example, given the reachability graph of diagram one above, a query might ask whether the following property is true of the system:
From every state with node_4 visible, there is a path whose next state has node_7 visible. This property

holds true of the example reachability graph.

In this simple example, states were labeled with propositions whose intended meaning regards the visibility of nodes at that state. More likely, we will not want to remember the node-number of each piece of information. We prefer to refer more meaningfully to the contents of a node, regardless of its node-number. We might, for example, assert that a state (collection of nodes) has proposition '*abstract*', that it gives a '*concrete example*', or that it includes an '*animation*'. These labels are essentially keywords.

Just as these properties of individual states may be descriptive, the dynamic properties of the whole system may be descriptive, and more useful than merely verifying the visibility of nodes. An example of such a dynamic property is: From every state with property *abstract*, there exists a path whose next state has property *concrete_example*. In hypertext synthesis, the user will specify this type of meaningful dynamic properties, and a hypertext will be synthesized which satisfies those properties.

Thus, we generalize M in our definition of a hypertext $< S_0, T, M >$. Rather than mapping a state to a set of visible nodes, it maps a state to a set of arbitrary propositions. Note that the visibility of nodes can still be asserted in this more general case, with propositions such as '*node_17*'.

3.3 Synthesis of Graphs

As recognized by Stotts et al. [21], property verification works *a posteriori*; after a finite state system has been created, we may verify whether that existing system satisfies certain dynamic properties. A complementary technique to property verification would work in the opposite direction; a user formulates a wish-list of properties he would like to see. Then a system is automatically created which manifests those properties. In this case, the desired properties come first -- *a priori* -- and the system is created to meet them. This is our approach to automatic synthesis of hypertext reachability graphs.

Our method of creating a reachability graph from a priori specifications uses an algorithm found in [3]. Using a temporal logic language 'UB', the user specifies -- a priori -- a formula of dynamic properties such as those in section 3.2 above. The authors detail a tableau-based decision procedure for satisfiability of UB formula. Their goal is to prove satisfiability of a formula, not to construct a model for use. However, as the tableau method is constructive, we adopt it to help construct our reachability graph. A model of UB formula is a graph, a branching structure. As Clarke et al. ([6], p. 68) point out "we may view the model as a flowgraph of global system behavior." We use this graph as the first step of constructing a hypertext reachability graph to satisfy the formula.

Section Four: Algorithm for Generating Hypertext Reachability Graph from User Specifications

4.1 Overview

For formal definitions and details of the algorithm, the reader is referred to the appendix. The approach is as follows: The basic algorithm found in [3] is extended to the special case of synthesizing hypertexts. The adapted algorithm has two stages and two inputs. The first stage is identical to that proposed in [3]. It takes as input the user's wish-list, and constructs a graph showing the propositions which must hold at each state and the required transitions between states. We call this a Constraint Reachability Graph (CRG), since it represents not a real hypertext reachability graph, but a depiction of constraints which must be met.

The algorithm's second stage takes as input the CRG and a hyperbase of nodes. The nodes are assumed to be labeled with meaningful keywords, as discussed in section 3.2. This second stage, then, attempts to populate the CRG with actual nodes whose labels match the CRG's labels at each state of the graph. The final result is a structure $H = <S_0, T, M>$ of actual, available hyperbase nodes, structured into a hypertext according to the original query's specifications. The entire process can be viewed as in figure one (next page).

We should clarify at this early point that our algorithm (only) synthesizes a reachability graph to show the visibility of nodes at each state and the potential browsing paths between states of the hypertext. In order to construct an actual hypertext with those properties, one must choose a data model (e.g. [20], [22]), and build a document which, under that model, will result in the desired reachability graph. For the sake of clearly presenting the essential synthesis method, our initial treatment is independent of the details of various data models; we focus on reachability graphs. The only essential detail we assume is that the hypertext data model provides some form of links, with which we implement the state transitions *T* of the reachability graph. Our concluding discussion suggests how recent advances in hypertext data modeling would be incorporated and leveraged for hypertext synthesis.

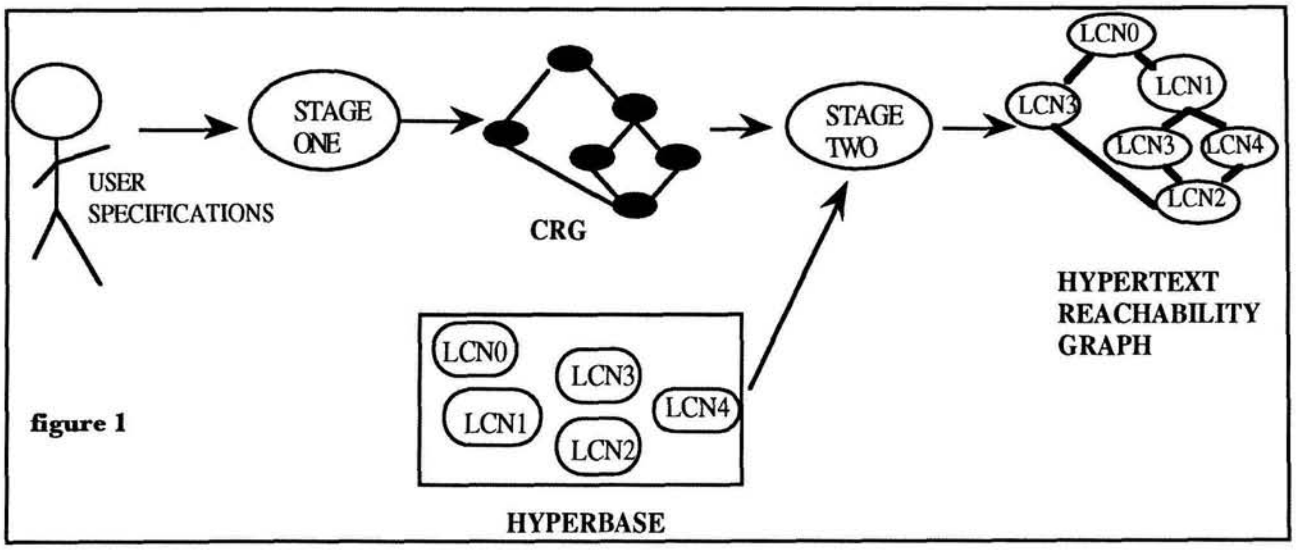


figure 1

4.2 Inputs

Hyperbase

There are two important points to be made about the input hyperbase. First, as discussed in section 1.2, only nodes are retrieved from the hyperbase; links are not retrieved. Our algorithm synthesizes new links between the retrieved nodes.

Second, when we speak of retrieving a node from the hyperbase, we actually intend, more generally, retrieving a *set* of nodes. Most hypertext data models allow for more than one node to be visible in each browsing state. Therefore, rather than labeling and retrieving individual nodes, we label and retrieve sets of nodes (of course, a state might include only a single node). We call these Labeled Collections of Nodes (LCN's).

Query Language

The second input is the user's wish-list, expressed in a query/specification language. We adopt Ben-Ari et al.'s straightforward UB language [3]. For an overview of temporal logic as it applies to hypertext reachability graphs, see [21]. In general, the language allows the users to express the properties (i.e. keywords) which must hold of the node (or set of nodes) at each state, as well as the browsing paths which must exist at each state.

4.3 Example

We will demonstrate all aspects of the algorithm using one running example of a researcher interested in data models for object-oriented databases. We may view the example as either a query, or as part of the specifications for a large pre-authored educational document. The example is admittedly simplified for the sake of clearly presenting the essential synthesis method. We introduce the example with an English version of a user's specification, followed by the UB-language equivalent.

English:
Initial state has property *date_model_headings* ;
On all paths, always, if a state has property *data_model_headings* then the next state has property *data_model_substance* ;
On all paths, always, if a state has property *query_language* then it has property *examples.*

UB:
data_model_headings
$\forall G$ (*~data_model_headings*
$\vee$ $\forall X$ *data_model_substnce*)
$\forall G$ (*~query_language* $\vee$ *examples*)

Stage One Output
Constraint Reachability Graph

One possible output of stage one is this CRG (next page):

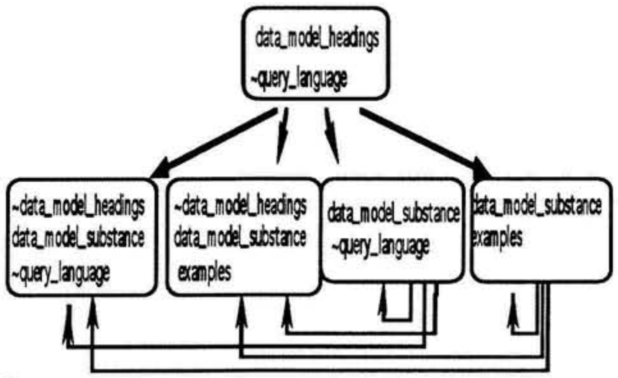


diagram two

This CRG shows the required content and browsing structure of the desired hypertext document. Note that it has two terminal states (i.e. states with no outgoing transitions) where browsing terminates. The intuition behind this termination is: Once a state is reached which has no headings for further readings, browsing ceases. The user might instead have specified, for instance, that one can always return to the start state, and our CRG would have had no terminal states.

A CRG exactly resembles a hypertext reachability graph. In fact, its formal definition is identical. A CRG is defined as a structure
CRG = $<CRGS_0$, T_CRG, M_CRG> where
CRGS is a set of states, P is a set of propositions and $CRGS_0 \in CRGS$
T_CRG: CRGS $\rightarrow 2^{CRGS}$
M_CRG: CRGS $\rightarrow 2^{P}$

The only difference between a CRG and our final output $<H = S_0$, T, M > is that the CRG is abstract; it only tells us which propositions must hold at each state. A second step must access an actual hyperbase and retrieve a set of nodes (LCN) to instantiate each CRG state.

There may be more than one possible CRG for a given input specification. In particular, if the specifications imply that the *start* state may have one of a few possible properties (a disjunction), then there will be one CRG for each start-state label. In our example, in fact, there exists an alternative CRG which differs only in the label on the start state. The number of alternatives is likely to be very small in general, but more work needs to be done to asses the empirical extent of this source of variability. The user may be presented with the alternatives, and he may select any or all for further processing in stage two.

Method of Stage One: Tableau

We present here only an overview of the method. Our appendix details the tableau procedure and applies it to our example. A root node is created and labeled with the user's specifications. A tree is then inductively created by applying transformation rules to its leaves. For example, for any label of the form '*p or q*', two child nodes are created, one of which is labeled '*p*', the other of which is labeled '*q*'. A tree is built in this fashion until certain conditions are met, when the procedure stops.

Stage Two: Populating the CRG with Actual LCN's

Stage two requires these two inputs:
1. One output CRG from stage one
2. the hyperbase of LCN's

This second stage is straightforward: For each state in the CRG, we search for an available LCN whose label satisfies the label on that state of the CRG. The only complication arises because the CRG states are labeled with simple formula of the form '*p*' **and** with negated formula of the form '*~p*'; the keyword labels on the LCN's, on the other hand, are labeled only with simple (non-negated) formula. An LCN satisfies the label of a CRG state if these two conditions hold:

1. For every simple formula '*p*' on the CRG state, the LCN is identically labeled '*p*'.
2. For every negated formula '*~p*' on the CRG state, the LCN is **not** labeled with '*p*'.

The result of this substitution of LCN's for CRG states is our final output: a hypertext reachability graph H = $<S_0$,T,M> constructed out of the available LCN's, which satisfies the original user specifications[3]:

In our example, suppose stage two were provided with the first CRG output of stage one (diagram 2), and with the following hyperbase of LCN's:

LCN1: *data_model_headings*
LCN2: *data_model_substance, beginner*
LCN3: *data_model_substance, examples, query_language, intermediate*
LCN4: *data_model_headings, data_model_substance, other*

Then one possible solution is this (next page):

[3] See appendix where we show more formally that the document *satisfies* the specifications using the UB semantics

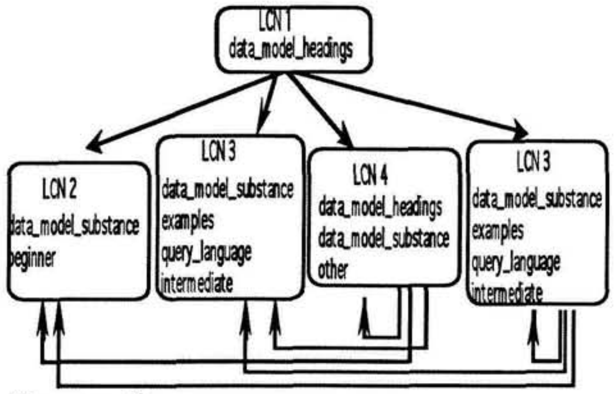


diagram three

4.4 Final Output: Discussion

The labels on the final instantiated hypertext states differ from those of the CRG: there are no labels of the form ~p, and there may be additional labels in each state which were not required by the CRG. The hypertext reachability graph is a concrete instantiation of the requirements, given the available hyperbase of information. The labels on the final output reachability graph show -- *for one concrete instantiation of the CRG* -- which propositions will actually hold at each state.

State five deserves attention. Recall that the specifications require that any state which has headings for further readings must be followed by substantive material. But in this particular instantiation of the CRG, state five, by chance, does not contain any headings -- yet it is still required to lead to substantive material. The 'mistake' here is in the specifications: To preclude this possibility, the specifications should also say that any state with *no* headings is *not* followed by substantive material.

Note that although states three and five have the same content -- i.e. the content of LCN3 -- they are distinct states because of the different outgoing transitions. In cases where two reachability graph states are instantiated with an identical LCN **and** have identical incoming and outgoing transitions, the reachability graph could be simplified by collapsing the two states into one.

Note also that there may be more than one possible substitution of LCN's for each CRG state. Considering all possible combinations, one CRG may represent many different reachability structures. There are, therefore, two sources of variability given a single specification; first, more than one CRG may result from stage one, if there is more than one possible starting state. Second, for each CRG, many different substitutions of LCN's for CRG states are possible.

This second source of variability can result in a large number of alternatives, if many LCN's in the hyperbase are identically labeled. In ordinary IR such as keyword search, we *expect* a list of resulting documents which match the keywords, and there is no problem. In our approach, the question is how to integrate these alternatives into a single hypertext structure. One solution is to create a list of the alternative LCN's for each CRG state. At each browsing state (corresponding to a CRG state), the user can scroll through the alternatives as he would if he were using traditional search methods. The formal definition of $H = < S_0, T, M>$ would have to be extended to $H = < S_0, T, M, L>$ where L is a mapping from each state S_i (accessible from S_0 by T) to a set of possible LCN's, or equivalently, to a set of possible sets of nodes visible in that state.

Ease of Maintenance

The separation of this algorithm into these two stages facilitates maintenance. In the case of pre-authored documents, the user's specifications are likely to remain fairly constant. On the other hand, the hyperbase may be constantly changing, as outdated information is replaced, and as new information becomes available. Fortunately, the abstract CRG does not rely on the existence of any particular node or LCN in the database. When the database is updated, only stage two needs to be re-executed to determine which of the currently-available LCN's should be used in the hypertext structure. This is especially fortunate because of the computational expensiveness of stage one, as discussed below.

Section Five: Extensions

For the sake of simplicity, we presented the algorithm assuming the simplest IR techniques, and largely independent of the details of hypertext data models. We now show how advances in the fields of IR and hypertext data modeling can be incorporated.

5.1 Keywords and IR

We have assumed the most primitive IR technique, keywords. Many of the more sophisticated approaches proposed for general text IR and for hypertext node retrieval are easily incorporated. Individual nodes (or collections of nodes) are retrieved using any of these more refined techniques; our synthesis algorithm focuses on constructing a browsing structure from among the retrieved components. So, for example, we may use attribute-value pairs as in [13] and relational queries as in [11] to reference individual components, without any modifications to our algorithm. In addition, if the nodes of the available LCN's happen to be incorporated into pre-authored hypertext documents, we might use the meaning of those links to help retrieve appropriate nodes, as in [10], [15].

5.2 Query/Specification Language

Admittedly, the UB language would be too formal for most users. A less formal language based on temporal logic would be required. But we anticipate that most users would experience difficulty not only with the formal syntax, but with adopting the mentality of branching logic. One cannot simply ask the user to supply structural qualities he wants, and expect that he will adopt the language of *paths* and *branches*. A user interface would need to guide him to assert browsing properties of the form "there exists a path, for all paths, next state, etc.". The current trend toward providing users with a gods'-eye-view of hypertext documents (to limit disorientation) should help acclimate users to these branching notions.

5.3 Nodes versus States

We simplified our presentation by requiring that states (LCN's) -- but not individual nodes -- are labeled. Our method is easily extended to allow labeling and retrieval of individual nodes, as shown immediately below. A query could then alternately reference *both* properties of a browsing state and properties of its constituent nodes. An example query is:

UB: $\forall G$ (p $\rightarrow$ ($\exists X$q and $\exists$n (r(n))))

English: Always, on all paths, from a state with property p there exists a path whose next state has state-property q and which has at least one node with (node) property r.

The algorithm would be extended as follows: Stage one, the hard part, simply labels a state in the CRG with q and with the complex label (En r(n)), i.e. at this state, q must hold and it must be true that there exists a node labeled r. Stage two, whose job it is to find LCN's to match the constraints of the CRG, is easily extended. It would take as additional inputs: the mapping M_nodes from individual nodes to their properties, and a new mapping V from each LCN to its set of visible nodes V: LCN $\rightarrow 2^N$. Armed with these two mappings, the second stage can easily find which LCNs satisfy the specification.

The more fundamental problem -- and this issue pertains to many areas of hypertext data model research, not only for automated synthesis -- is how to relate the contents and labels of individual nodes to the labeling of LCN's which contain those nodes. We assumed above that simple labels at least had the advantage of being automatically generated. But this assumption overlooks the question of how to automatically generate keyword labels for an LCN which is a set of nodes. Are they merely a conjunction of all the keywords we would generate for each node in the LCN ? Would we not want other labels to describe the more general idea of the *set* of nodes, or properties like "*half* the nodes in this LCN are graphic" ? Garzotta et al. ([13]) raise this difficult issue with respect to labeled links, in the case of a hierarchical data model. These questions arise in all cases where we label components at multiple levels of abstraction. They are left open for future work.

5.4 Algorithmic Complexity and Data Abstraction

The complexity of the satisfiability problem is exponential in the length of the formula. This is bad news for our otherwise straightforward approach. The problem is likely to be much worse when our synthesis methods used to automatically generate a pre-authored document than when it is used as a query for information retrieval, simply because of the size of the formula. Fortunately, most pre-authored documents have a fair amount of hierarchy, as prescribed in [16] and [1]. The specifications can be written, and a reachability graph synthesized, one level at a time. For example, in authoring a tour-book, we may specify the browsing properties between countries ("from every country there exists a path to its immediate neighbors"), then between cities, etc., all the way down, say, to the browsing properties among restaurants within a city. The overall reachability graph is thus built top-down. This is only feasible, of course, where the author in fact wants such a strong hierarchy. In future work we will elaborate on how to utilize data models which support hierarchical composition (e.g. [5]) to help alleviate the potential exponential complexity of our algorithm.

5.5 Links

In the case of property *verification*, we can verify that a state has visible a certain named link (e.g. link_17), as in [21]. In a more sophisticated model, we could verify that a link of some type is visible. Furthermore, we can label links with arbitrary properties, and by extending the specification language as we did for individual nodes, we could quantify over individual links and verify their properties.

But the case of hypertext *synthesis* is much more complicated. Allowing specification of links in addition to nodes and states would require a logic of links within the synthesis algorithm. For example, what is the meaning of specifying that a state has no successors but that it has a link, or that it has a successor of some type but no links which (by their link-type definition) can lead to such a node, and other such contradictions ? In our proposed synthesis algorithm these contradictions are avoided, since the requirement of links is left *implicit* in the requirements to have paths between states. To allow explicit specification of states, paths among states, *and links* requires a logic of these components that understands how they relate. Such a logic is left for future work.

Conclusions

In this paper we have proposed a method to automatically synthesize a hypertext reachability graph from temporal logic specifications. The method is an extension of Ben-Ari's algorithm to the case of hypertext. This technique serves two important uses. First, for the case of pre-authored hyperdocuments, a structure is automatically synthesized which necessarily meets the author's specifications. This takes the guesswork out of manual link-creation, in which the author must envision how each new link will affect the browsing structure and its properties. More importantly, we propose that these specifications -- viewed as queries -- serve as a powerful IR technique. Individual nodes of information are precisely targeted and combined into a document ready for browsing. With today's limited technology, the user himself must include structural properties in his query. At some later time, an intelligent technique may specify those properties automatically. Either way, our algorithm retrieves nodes and sets of nodes and produces among them a browsing structure which satisfies those properties. The user then has all the relevant information he needs in the desirable form of a tailor-made hypertext document.

References

1. Acksyn, Robert M., McCracken, Donald L., and Yoder, Elise A. KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. In *Communications of the ACM*, vol. 31, no. 7, July 1988, pp. 820-835

2. Beeri, Catriel and Kornatzky, Yoram. A Logical Query Language for Hypertext Systems. In *Proceedings of the European Conference on Hypertext*, Camridge University Press, France, November 1990, pp. 67-80

3. Ben-Ari, M., Manna, Z., and Pnueli, A. The temporal Logic of Branching Time. In *Proceedings 8th ACM Symposium on Principles of Programming Languages*, ACM, New York, 1981, pp. 164-176

4. Bieber, Michael. Providing Information Systems with Full Hypermedia Functionality. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, IEEE, vol. 3, January 1993, pp. 390-400

5. Casanova, Marco A. and Tucherman, Luiz. The Nested Context Model for Hyperdocuments. In *Hypertext '91 Proceedings*, ACM Press, December 1991, pp. 193-201

6. Clarke, Edmund M. and Emerson, E. Allen. Synthesis of Synchronization Skeletons from Branching Time Temporal Logic. In *Proceedings of the Workshop on Logics of Programs* (Yorktown-Heights, New York). Lecture Notes in Computer Science, Springer-Verlag, New York, 1981, pp. 52-71

7. Clarke, E.M., Emerson, E.A., and Sistla, A.P. Automatic Verification of Finite-State Concurrent Systems Using Tempoal Logic Specifications. In *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, April 1986, pp. 244-263

8. Consens, Mariano P. and Mendelzon, Alberto O. Expressing Structural Hypertext Queries in Graphlog. In *Hypertext-89 Proceedings*, ACM, November 1989, pp. 269-292

9. Egan, Dennis E., Remde, Joel R., Gomez, Louis M., Landauer, Thomas K., Eberhardt, Jennifer, and Lochbaum, Carol C. Formative Design-Evaluation of Superbook. In *ACM Transactions on Office Information Systems*, vol. 7, no. 1, January 1989, pp. 30-57

10. Frei, H.P. and Stieger, D. Making Use of Hypertext Links when Retrieving Information. In *Proceedings of the ACM Conference on Hypertext*, November 1992, pp. 102-111

11. Gallagher, Leonard, Furuta, Richard, and Stotts, P. David. Increasing the Power of Hypertext Search with Relational Queries. In *Hypermedia*, vol. 2, no. 1, 1990, pp. 1-14

12. Garrett, Nancy L., Smith, Karen E., and Meyrowitz, Norman. Intermedia: Issues, Strategies, and Tactics in the Design of a Hypermedia Document System. Institute for Research in Information and Scholarship (IRIS), Brown University, Providence, RI.

13. Garzotto, France, Mainetti, Luca, and Paolini, Paolo. Navigation Patterns in Hypermedia Data Bases. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, IEEE, vol. 3, January 1993, pp. 370-379

14. Garzotto, Franca, Paolini, Paolo and Schwabe, Daniel. HDM - A Model-Based Approach to Hypertext Application Design. In *ACM Transaction on Information Systems*, January '93, pp. 1-26

15. Guinan, Catherine and Smeaton, Alan F. Information Retrieval from Hypertext Using Dynamically Planned Guided Tours. In *In Proceedings of the ACM Conference on Hypertext*, November 1992, pp. 122-130

16. Halasz, Frank G. Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. In *Communications of the ACM*, vol. 31, no. 7, July 1988, pp.836-852

17. Hua, Hua and Kimbrough, Steven O. On Hypermedia-Based Argumentation Decision Support Systems. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, IEEE, vol. 3, January 1993, pp. 401-410

18. Lucarella, Dario. A Model for Hypertext-Based Information Retrieval. In *Proceedings of the European Conference on Hypertext*, Cambridge University Press, France, November 1990, pp. 81-94

19. Shasha, Dennis. NetBook - a data model to support knowledge exploration. Technical Report, Department of Computer Science, Courant Institute of Mathematical Sciences, New York, 1985

20. Stotts, P. David and Furuta, Richard. Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. In ACM Transactions on Information Systems, vol. 7, no. 1, January 1989, pp. 3-29

21. Stotts, P. David, Furuta, Richard, and Ruiz, J. Cyrano. Hyperdocuments as Automata: Trace-Based Browsing Property Verification. In *Proceedings of the ACM Conference on Hypertext*, November 1992, pp. 272- 281

22. Tompa, Frank WM. A Data Mode for Flexible Hypertext Database Systems. In *ACM Transaction on Information Systems*, vol. 7, no. 1, January 1989, pp. 85-100

23. Wilson, Eve. Links and structures in hypertext databases for law. In *Proceedings of the European Conference on Hypertext*, Cambridge University Press, France, November 1990, pp. 194-211

Appendix

Semantics of UB

The semantics of a UB formula are defined with respect to a tree: A formula is satisfied at a state *s* of a tree *T*. In our case, we want the specifications to be satisfied at the state S_0 of the infinite tree represented by the hypertext reachability graph.

The following excerpt from [3] formally defines the semantics of UB formula:

> "A model T for UB is a triple T = (S,P,R) where S is a set of states and P is an assignment of proposition letters to states. For a proposition a and a state $s \in S$, $a \in P(s)$ iff a is true at the state s. R is a binary relation on states which defines the structure of T. When sRt holds, we say that t is an immediate successor (descendant) of s. To capture the concept of non-ending time, we require that R be total, i.e. forall s there-exists t s.t. (sRt) - every state has a successor[1] . R* is the reflexive transitive closure of R. Thus sR*t iff there is an R-path leading from s to t. An s-branch b is an infinite path $b = (s = s_0, s_1, \ldots)$ such that $s_i \in S$ and $s_i R s_{i+1}$. We define the notion of a general formula p being satisfied at a node s in T - written as $T, s \models p$ or $s \models p$ when T is implicitly understood.
>
> 1. For a proposition a, $s \models a$ iff $a \in P(s)$.
> 2. $s \models \sim p$ iff $s \not\models p$
> 3. $s \models p \vee q$ iff $s \models p$ or $s \models q$
> 4. $s \models \forall G p$ iff $\forall b \forall t (t \in b \rightarrow t \models p)$
> 5. $s \models \forall F p$ iff $\forall b \exists t (t \in b$ and $t \models p)$
> 6. $s \models \forall X p$ iff $\forall t (sRt \rightarrow t \models p)$
> 7. $s \models \exists G P$ iff $\exists b \forall t (t \in b \rightarrow t \models p)$
> 8. $s \models \exists F p$ iff $\exists b \exists t (t \in b$ and $t \models p)$
> 9. $s \models \exists X p$ iff $\exists t (sRt$ and $t \models p)$"

In this paper, our aim is to synthesize a structure $H = <S_0, T, M>$ (this triple differs notationally from Ben-Ari's in our choice to include only S_0, not S) which satisfies the specifications "Spec". Utilizing Ben-Ari's precise semantics, H satisfies "Spec" iff
$H, S_0 \models$ "Spec".

Ben-Ari's Tableau Method for UB

The method constructs a tree T out of the initial specification. The root of T is created and labeled with the originial specifications. The tree T is then constructed by inductively applying these rules to its leaves: Formula in the leaf are matched against the α column of table 1.a and the β column of table 1.b. For every formula which matches an α pattern, a child node is created which is labeled by the corresponding α_1 and α_2. For every formula which matches a β pattern, two child nodes are created, one of which is labeled with the corresponding β_1 , the other of which is labeled with β_2. When no matches can be found for a leaf's formula, that leaf is designated an X-node. If a node contains two contradictory formula, one of the form 'p' and one of the form '~p', the that node is not expanded any further.

[1] Although Ben-Ari defines the meaning of formula with respect to an infinite structure representing the non-ending nature of time, his algorithm for constructing a model from specifications allows construction of finite graphs with terminal states. In fact, our simple example with which we demonstrate our extension to his algorithm synthesizes a finite hypertext with terminal states. We can either loosen the definition of a model to allow finite time, or we can extend the algorithm to arbitrarily add a path from the terminal nodes to non-terminal nodes.

α	$\alpha 1$	$\alpha 2$
$p \wedge q$	p	q
$\sim(p \vee q)$	$\sim p$	$\sim q$
$\sim\sim p$	p	p
$\forall G p$	p	$\forall T \forall G p$
$\exists G p$	p	$\exists T \exists G p$
$\sim\forall\left\{\begin{matrix}G\\F\\X\end{matrix}\right\}p$	$\exists\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$	$\exists\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$
$\sim\exists\left\{\begin{matrix}G\\F\\X\end{matrix}\right\}p$	$\forall\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$	$\forall\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$

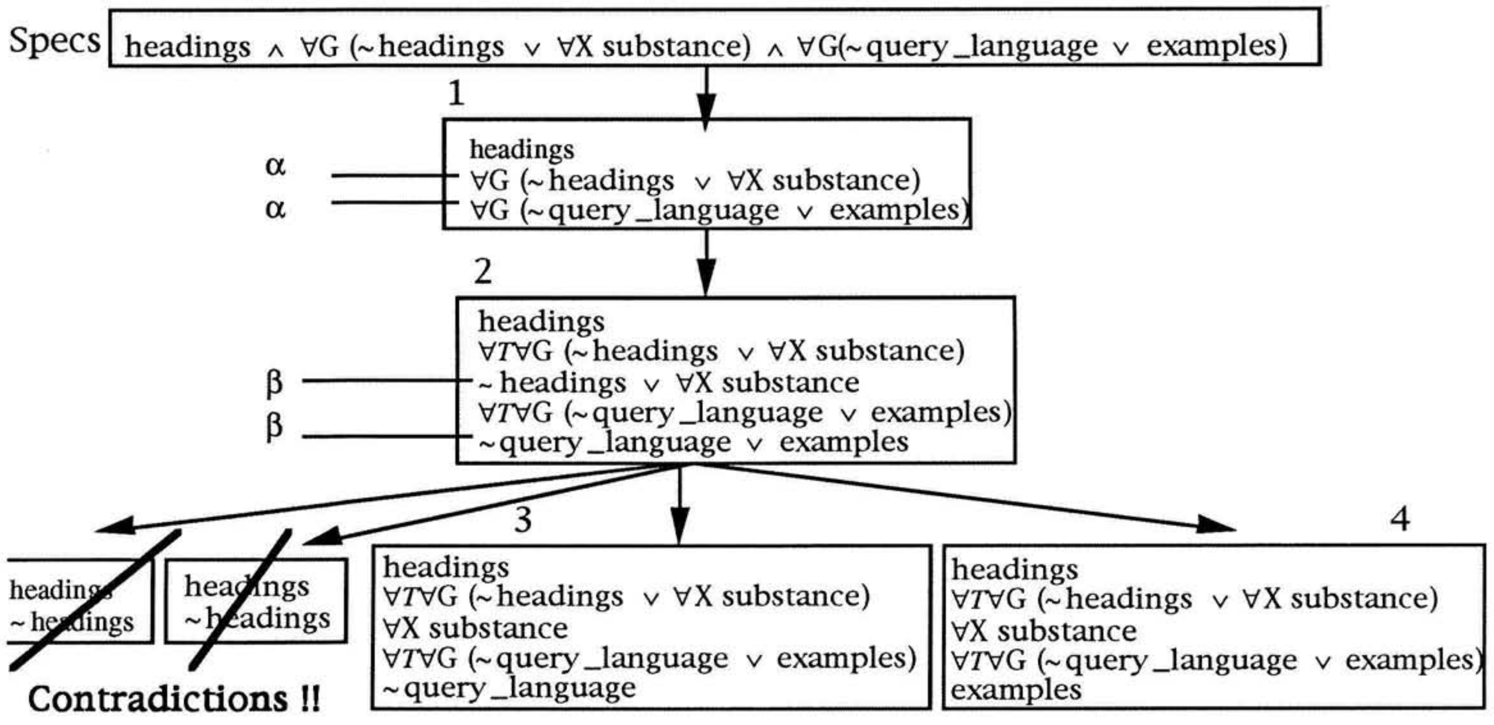
table 1.a

β	$\beta 1$	$\beta 2$
$p \vee q$	p	q
$\sim(p \wedge q)$	$\sim p$	$\sim q$
$\exists F p$	p	$\exists X \exists F p$
$\forall F p$	p	$\forall X \forall F p$

table 1.b

We will demonstrate this first part of the algorithm using our running example. As a shortcut, we simultaneously apply our rules to more than one subformula in each iteration. This shortcut does not affect the final result. Also, for brevity, we use 'headings' in place of 'data_model_headings' and 'substance' in place of 'data_model_substance':

Specifications:
headings ∧ ∀G (~headings ∨ ∀X substance) ∧ ∀G (~query_language ∨ examples)


Specs
headings ∧ ∀G (~headings ∨ ∀X substance) ∧ ∀G(~query_language ∨ examples)
1
α
α
headings
∀G (~headings ∨ ∀X substance)
∀G (~query_language ∨ examples)
2
β
β
headings
∀T∀G (~headings ∨ ∀X substance)
~headings ∨ ∀X substance
∀T∀G (~query_language ∨ examples)
~query_language ∨ examples
headings
~headings
headings
~headings
Contradictions !!
3
headings
∀T∀G (~headings ∨ ∀X substance)
∀X substance
∀T∀G (~query_language ∨ examples)
~query_language
4
headings
∀T∀G (~headings ∨ ∀X substance)
∀X substance
∀T∀G (~query_language ∨ examples)
examples

At this point, we have only two X-nodes which contain no contradictions. X-nodes contain simple formula of the form '*p*' or '*~p*', and *nexttime* formula of the form
$\forall X p$ and $\exists X p$.[2] The processing for X-nodes is most easily expressed using Ben-Ari's notation. Suppose the set of nexttime formula is this: $\{\exists X_{p1},\ldots \exists X_{pk}, \forall X_{q1},\ldots \forall X_{qm}\}$. Then, for each formula of the form $\exists X pi$, a child is created and labeled { pi, q1,..,qm} . For all the formula $\forall X q1,.., \forall X qm$, one child is created and labeled q1..qm.

> "The construction of T is kept finite by observing the following two termination rules:
> T1: If a created node n contains both p and ~p then mark this node as closed and do not expand it any further.
> T2: If a state m is to be created as a son of n, and there is a state t (which has already been created) elsewhere in the tableau [with identical labels] ...then do not create m but connect n to t instead."

Two more steps conclude the tableau method. We will only briefly describe these two steps, and refer the reader to [3]. First, we delete a node if any of the following apply:

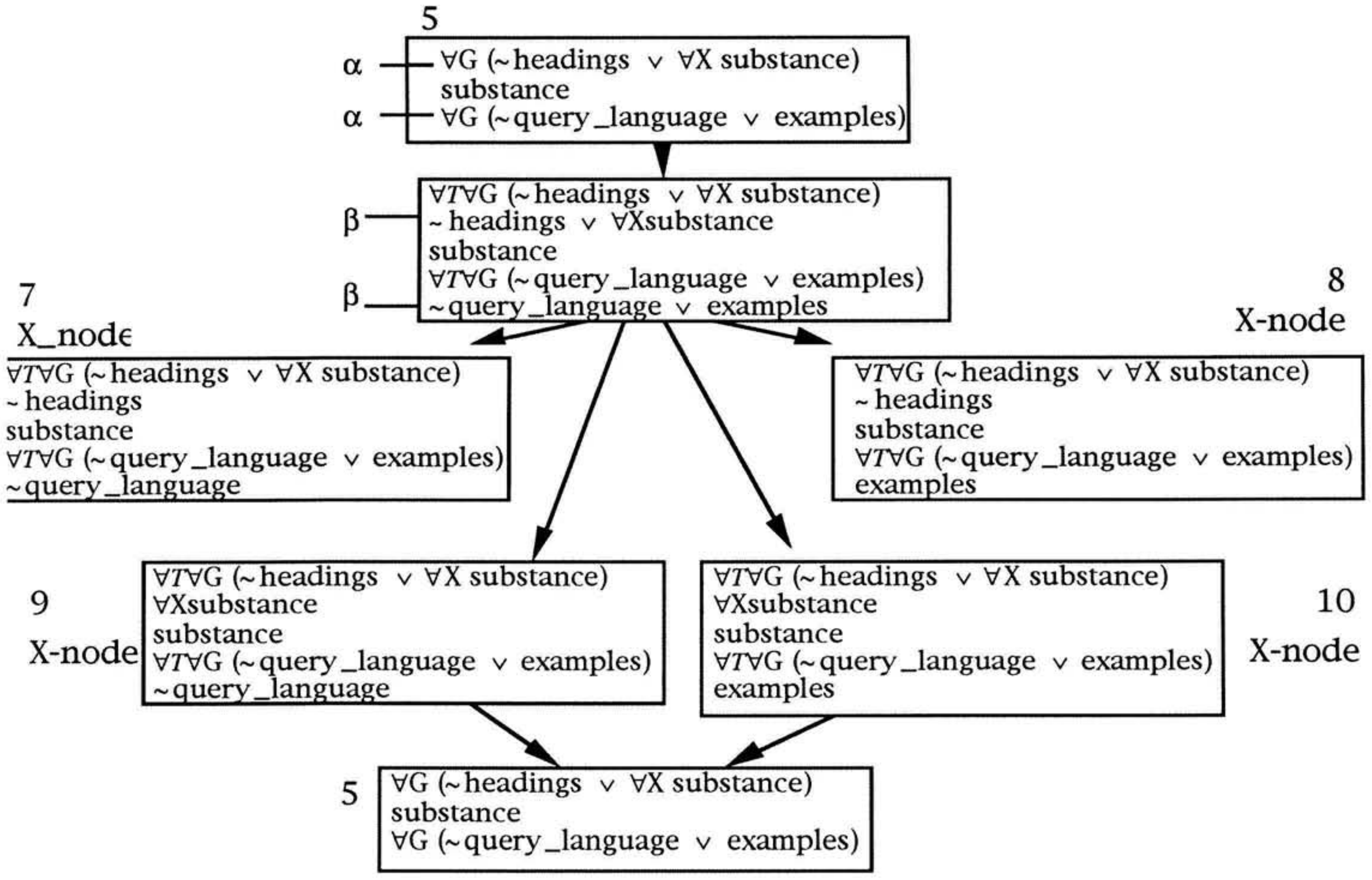
its labels assert both p and ~p
it is an α node and one of its sons is deleted
it is a β node and both its sons are deleted
it is an X-node and one of its descendants has been deleted
it is a node labeled $\exists F$p and there exists no path from the node to a state labeled p
it is a node labeled $\forall F$p and there exists a path along which p cannot be fulfilled

This last rule is deceptively tricky. We do not delete the $\forall F$p node if there merely exists a path along which p is not fulfilled. The construction of T actually creates such cycles, since for each occurence of $\forall F$p, we provide a son labeled $\forall X \forall F$p as well as the son labeled p. By cycling in a path labeled $\forall X \forall F$p, we can forever avoid fulfilling p. Instead, this last rule only deletes a node labeled $\forall F$p if there exists a path along which p **cannot** be fulfilled.

For all other $\forall F$p nodes, every path can eventually fulfill p. In a final step, we modify the tree to ensure that every such path does *in fact* fulfill p, by unraveling the tree to avoid the possibility of a cycle of $\forall X \forall F$p. With this overview of this last step in mind, the reader is refered to [3].

We now return to our example and continue building the remainder of the tree:

[2] Formula of the form $\forall T f$ (or $\exists T f$) are just a method to propogate the truth of $\forall G f$ ($\exists G f$) to all states. Others ([8]) have taken a less formal approach to these formula, since they clutter-up the tableau method. Ben-Ari does not explain how he treats these $\forall T f$ ($\exists T f$). To complete the formal approach to these formula, we add this to Ben-Ari's rules:
If a node is an X-node node with a *nexttime* formula, then treat $\forall T f$ ($\exists T f$) as $\forall X f$ ($\exists X f$) Otherwise it does not call for any expansion.


5
α
∀G (~headings ∨ ∀X substance)
substance
α
∀G (~query_language ∨ examples)
β
∀T∀G (~headings ∨ ∀X substance)
~headings ∨ ∀Xsubstance
substance
∀T∀G (~query_language ∨ examples)
β
~query_language ∨ examples
7
X_node
∀T∀G (~headings ∨ ∀X substance)
~headings
substance
∀T∀G (~query_language ∨ examples)
~query_language
8
X-node
∀T∀G (~headings ∨ ∀X substance)
~headings
substance
∀T∀G (~query_language ∨ examples)
examples
9
X-node
∀T∀G (~headings ∨ ∀X substance)
∀Xsubstance
substance
∀T∀G (~query_language ∨ examples)
~query_language
10
X-node
∀T∀G (~headings ∨ ∀X substance)
∀Xsubstance
substance
∀T∀G (~query_language ∨ examples)
examples
5
∀G (~headings ∨ ∀X substance)
substance
∀G (~query_language ∨ examples)

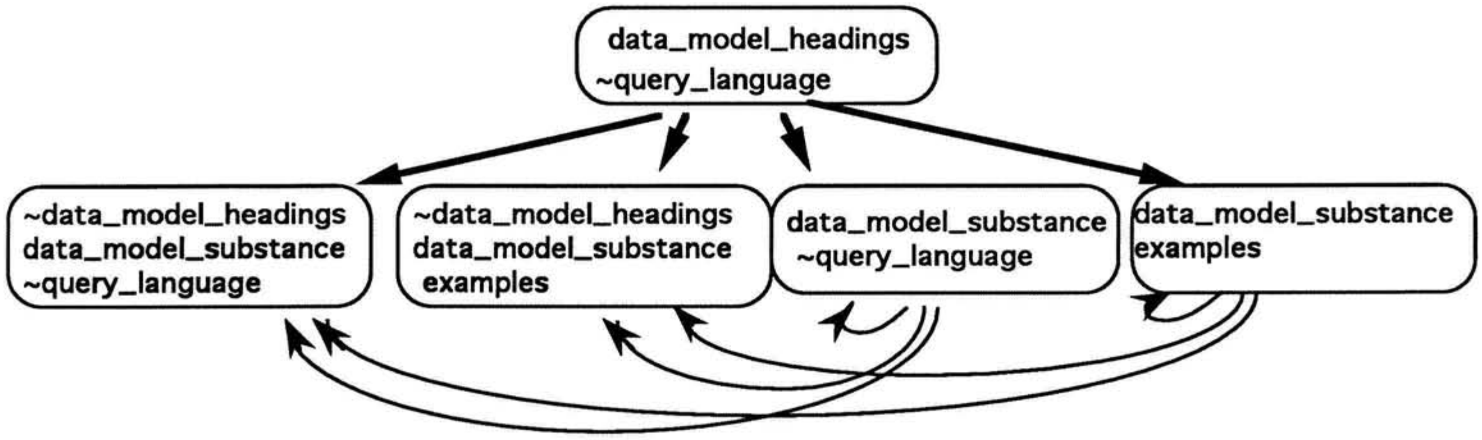
The last node is identical to node five, so nodes nine and ten are connected to node five and processing is complete. This graph is not yet the final output CRG of stage one. The CRG must be extracted from this graph.

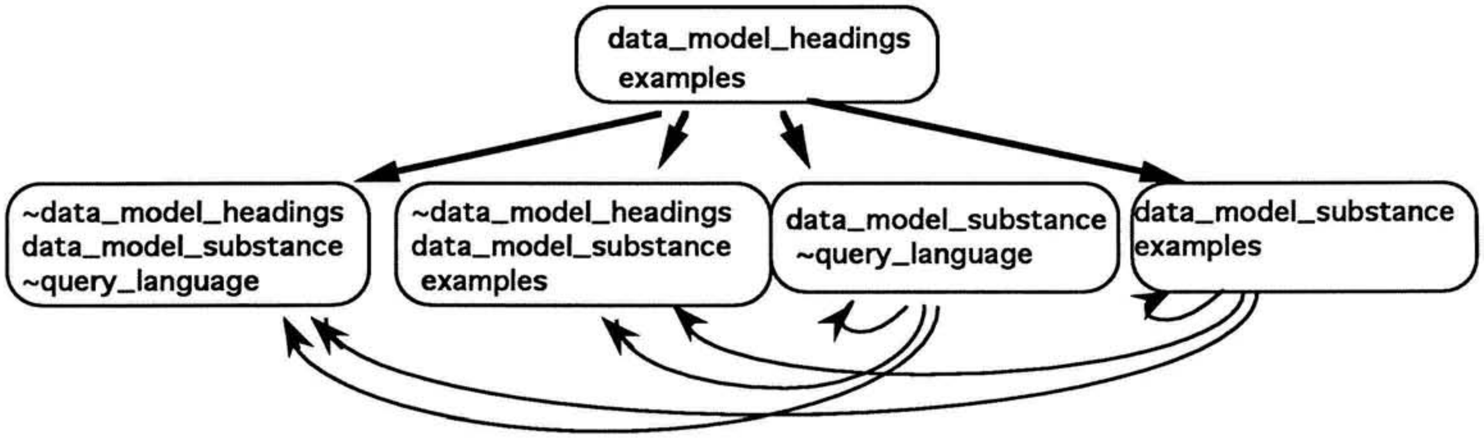
Extracting the CRG from T

Every X-node is defined as a state. It is these 'states' in the structure T which represent fragments of the model for p which we seek. The other nodes of T are merely used to help compute the solution. We extract the model of p from the tree T by retaining only those nodes which are 'states' -- i.e. X-nodes. The transition relation is derived as follows: s_1Rs_2 if s_1 is encountered in T at depth i, s_2 is a descendant of s_1 and is encountered at depth j, and no descendant of s_1 is encountered at depth $k < j$.

In each retained state we retain only elementary labels of the form 'p' or '~p'. S_0 of the CRG is the first state encountered when traversing T breadth-first. If more than one state is encountered at the same depth of T, one is chosen arbitrarily. We may construct a CRG for each possible choice, to offer the user alternative solutions from which to choose. From that point, we traverse T, retaining only the states.

In our example, by retaining only the 'states' of T as described, we obtain these two simple CRG's, one for each possible choice of S_0. In general, a different selection of S_0 can result in a very different graph. In this simple case, only S_0 differs. For readability, we replace our abbreviated 'headings' with 'data_model_headings' and 'substance' with 'data_model_substance':





Stage Two
Populating the CRG with Actual LCN's

In this second stage, we find an actual LCN to populate each state in the CRG. When we find an LCN for each CRG state, we have built one model of the original specifications.

Ben-Ari et al. [3] show that the CRG satisfies the specifications. We now show that one instance of the CRG -- populated by LCN's in the manner described -- also satisfies the original specifications.

The only differences between the labels of the CRG and the labels of the final Hypertext Reachability Graph (HRG) are:
1. The CRG has negated labels of the form '~p'. But by the semantics of UB, a formula '~p' is satisfied iff the state is not labeled 'p'. And by construction, the HRG state is guaranted not to be labeled 'p'.

2. The HRG states may be labeled by extra propositions about which the CRG and the original formula asserted neither 'p' nor '~p'. The presence of these extra propositions is irrelevant to the semantics of UB.