Navigation in Hypertext is Easy Only Sometimes

Mark Levene
University College London
Gower Street
London WC1E 6BT, U.K.
email: mlevene@cs.ucl.ac.uk

George Loizou
Birkbeck College
Malet Street
London WC1E 7HX, U.K.
email: george@dcs.bbk.ac.uk

May 21, 1999

Abstract. One of the main unsolved problems confronting Hypertext is the navigation problem, namely the problem of having to know where you are in the database graph representing the structure of a Hypertext database, and knowing how to get to some other place you are searching for in the database graph. In order to tackle this problem we introduce a formal model for Hypertext. In this model a Hypertext database consists of an information repository, which stores the contents of the database in the form of pages, and a reachability relation which is a directed graph describing the structure of the database. The notion of a trail, which is a path in the database graph describing some logical association amongst the pages in the trail, is central to our model.

We define a Hypertext query language for our model based on a subset of propositional linear temporal logic, which we claim to be a natural formalism as a basis for establishing navigation semantics for Hypertext. The output of a trail query in this language is the set (which may be infinite) of all trails that satisfy the query. We show that there is a strong connection between the output of a trail query and finite automata in the sense that, given a Hypertext database and a trail query, we can construct a finite automaton representing the output of the query, which accepts a star-free regular language. We show that the construction of the finite automaton can be done in time exponential in the number of conjunctions, between the subformulae of the trail query, plus one.

Given a Hypertext database and a trail query, the problem of deciding whether there exists a trail in the database that satisfies the trail query is referred to as the model checking problem. We show that, although this problem is NP-complete for different subsets of our query language, it can be solved in polynomial time for some significant special cases. Thus the navigation problem can only be efficiently solved in some special cases, and therefore in practice Hypertext systems could include algorithms which return randomized and/or fuzzy solutions.

Key words. Hypertext, navigation, trail query, temporal logic, finite automata, computational complexity

AMS(MOS) subject classification. 68P15, 68P20, 68Q15

1   Introduction

Traditional text, for example in book form, has a single linear sequence defining the order in which the text is to be scanned. In contrast Hypertext [CONK87] (or more generally Hypermedia, see [HALA88]) is text (which may contain multimedia) that can be read nonsequentially. Hypertext presents several different options to readers, and the individual reader chooses a particular sequence at the time of reading.

The inspiration for Hypertext comes from the *memex* machine proposed by Bush [BUSH45] (see also [NYCE89]). The memex is a "sort of mechanized private file and library" which supports "associative indexing" and allows navigation whereby "any item may be caused at will to select immediately and automatically another". Bush emphasizes that "the process of tying two items together is an important thing". In addition, by repeating this process of creating links we can form a *trail* which can be traversed by the user, in Bush's words "when numerous items have been thus joined together to form a trail they can be reviewed in turn". Hypertext can be viewed as the formalization and realization of Bush's original ideas.

A *Hypertext database* [STOT89, TOMP89, FRIS92] is formalized as a directed graph [BUCK90] (called the *database graph*) whose nodes represent textual units of information (called *pages*) and whose arcs (also called *links*) allow the reader to navigate from an anchor node to a destination node. The structure of a Hypertext database changes over time, and thus differs from traditional databases, such as relational databases [ULLM88], which have a regular structure defined by a relational database schema. It is evident that a Hypertext database can be implemented on top of a relational database provided it has the facilities to store and retrieve textual objects. A comparison of the functional characteristics of several Hypertext systems can be found in [SCHN88].

The process of traversing links and following a *trail* of information in a Hypertext database is called *navigation* (or alternatively *link following*). In graph-theoretic terms a trail in a Hypertext database is a *path* in the database graph (we allow a node to occur more than once in a path; paths are called *walks* in [BUCK90]).

Navigating through a Hypertext database leads to the problem of getting "lost in hyperspace" [CONK87, VAND88], which is the problem of having to know where you are in the database graph and knowing how to get to some other place that you are searching for in the database graph. From now on we will refer to this fundamental problem as the *navigation problem*.

In order to solve the navigation problem we can augment link following with a *query*-based access mechanism [HALA88]. Such a mechanism allows users to specify the characteristics of the information they are searching for and then to obtain the output of their query from the Hypertext system. Two types of querying mechanism are:

- *Content-based search*, which typically uses index-based information retrieval technology. When searching by content, pages are searched independently of their association with other pages in the database.

- *Structure-based search*, which extends content-based search by additionally specifying a description of a subgraph [BUCK90] of the database graph. When searching by structure, contents of pages are searched in association to their linkage with other pages in the database according to the said specified subgraph.

We will refer to the formal semantics of the query mechanism used in a Hypertext system as its *navigation semantics*.

In a more general context of a Hypertext system the process of finding and examining the text pages associated with destination nodes is called *browsing* [CONK87]. The *browser* is the component of a Hypertext system that helps users search for the information they are interested in by graphically displaying the relevant parts of the database and providing contextual and spatial cues with the use of *navigational aids* such as *maps* and *guides* [FRIS92]. A set of tools that aid navigation by performing a structural analysis of the database graph is described in [RIVL94].

Herein, we are only interested in the navigation semantics of structure-based search as a means of attempting to solve the navigation problem and thus assume that navigational aids are available within the browser and are independent of the navigation semantics.

Example 1 The database graph of a Hypertext database, which models a simple *Teletext* system storing pages of information according to the following topics: news, sports, travel and weather, is shown in Figure 1. In a more comprehensive example the nodes in this database graph can be
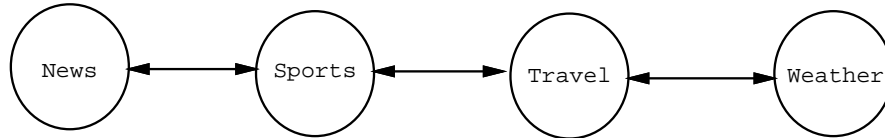
Figure 1: A simple database graph

expanded into subgraphs, for instance the node labeled news can be expanded into several pages one for regional news, one for country-wide news, one for European news and one for world-wide news.

In our Hypertext model we separate the contents of nodes (i.e. the pages) from the database graph. We define a Hypertext database to be an ordered pair $<I, R>$, where I is an *array* [TREN73] of pages, called the *information repository*, and R is a binary relation in the indices of I, called the *reachability relation*; the indices of I are referred to as the *page numbers* of I. Thus arrays are used to provide an indexing mechanism for pages. This leads us to define a *trail* as an array of page numbers corresponding to a path in the directed graph representing R.

In the navigation semantics that we propose, the notion of a trail is central. The output of a query is the set of trails in the Hypertext database satisfying the query specification; each trail in the output of a query is called an *answer* of the query. Therefore, a query consists of two orthogonal parts:

(i) logical formulae composed of predicates to be satisfied by the contents of pages in the information repository, and

(ii) the order in which the pages, satisfying the logical formulae, are to be found in the reachability relation.

An important requirement of the query language is that users should have as much flexibility as possible when posing a query according to their knowledge of the contents and structure of the Hypertext database.

This suggests that a logic of *positions* [RESC71] would be well suited as a query language for Hypertext due to the spatial interpretation of the database graph. As pointed out by Rescher and Urquhart [RESC71] there is a close connection between *positional* and *temporal* logic. Therefore we choose to utilize *propositional linear temporal logic* (PLTL) [EMER90] as the underlying semantics for navigation, where in our context *time* actually means *position*. (We mention that temporal logic is widely employed in the very active research area of specification and verification of concurrent programs [EMER90, MANN92].)

In our context we view a Hypertext database as a *temporal structure* [EMER90], where the page numbers of I are associated with the states of the structure and the binary relation R is associated with the transition relation of the structure.

We define a query language, called *Hypertext Query Language* (HQL), based on a subset of PLTL and call HQL formulae *trail queries* (or HQL queries). In particular, HQL includes the temporal operators "nexttime" (denoted by $\bigcirc$) and "sometimes" (denoted by $\Diamond$). A fundamental difference between the semantics of PLTL and HQL is that models of PLTL formulae are *timelines* which are infinite paths describing computation sequences [EMER90, THOM90], while models of HQL formulae are *trails* which are finite paths describing sequences of pages that can be traversed by users. Thus, in addition to the temporal operators $\bigcirc$ and $\Diamond$, HQL supports a novel temporal operator "finaltime" (denoted by $\triangle$), which refers to the final page number in a trail. (We will show that $\triangle$ adds expressive power to the query language.)

HQL provides a natural vehicle for combining content-based querying with structure-based search via the above temporal operators. The "nexttime" operator allows the user to navigate

through the pages one step at a time, the "sometimes" operator allows the user to skip as many pages as necessary in order to arrive at the next page he wishes to browse through, and the "finaltime" operator allows the user to specify the final page in a navigation session.

The output of a trail query is defined to be the set of all trails that satisfy the query, and the problem of whether there exists an answer to a trail query with respect to a Hypertext database is the problem of deciding whether the database is a model of the trail query; this problem is known as the *model checking* problem (see [EMER90]).

In order to construct the output of a query we take advantage of the strong connection between finite automata [HOPC79, PERR90] and PLTL [VARD86a, VARD86b, EMER90, THOM90] as follows.

Firstly, we show that a Hypertext database, say H, can be represented by a finite automaton $\mathcal{M}_H$ and that the language accepted by $\mathcal{M}_H$ is a *star-free* regular language [MCNA71, PERR90]. Secondly, we show that a finite automaton $\mathcal{M}_f$ can be constructed for a trail query, say f, with respect to the Hypertext database H, and that the language accepted by $\mathcal{M}_f$ is also a star-free regular language. Finally, we show that the output of a trail query corresponds to the intersection of $\mathcal{M}_H$ and $\mathcal{M}_f$, which also accepts a star-free regular language, since star-free regular languages are closed under intersection. Thus the navigation semantics of HQL are given in terms of star-free regular languages.

In addition, we show that constructing $\mathcal{M}_H \cap \mathcal{M}_f$ can be done in time exponential in the number of conjunctions, between the subformulae of f, plus one, and in the case when the number of conjunctions between the subformulae of f is one, then the construction can be done in time polynomial in the size of H and the length of f. Thus, we give an exponential time upper bound for query evaluation in HQL, which is tractable as long as the number of conjunctions in the subformulae of trail queries is bounded by some constant.

In order to measure the difficulty of navigation in Hypertext we investigate the computational complexity of the model checking problem for HQL queries. We prove that in the general case the model checking problem is NP-complete, which is to be expected by inspecting the results in [SIST85] (see also [EMER90]). We also show that in two special cases, when trail queries do not contain any occurrences of $\diamond$, the model checking problem can be solved in polynomial time in the length of the information repository I and the length of the trail query f.

The rest of the paper is organized as follows. In Section 2 we briefly survey related work. In Section 3 we formalize the notion of a Hypertext database. In Section 4 we present our query language, HQL. In Section 5 we investigate a strong connection between the output of a trail query and finite automata that accept star-free regular languages. In Section 6 we investigate the computational complexity of navigation in a Hypertext database. Finally, in Section 7 we give our concluding remarks.

2   A Brief Survey of Related Work

Several researchers [STOT89, STOT92, MEND95] have recognized that the semantics of navigation in a Hypertext database can be formalized in terms of finite automata [HOPC79, PERR90].

In [MEND95] a query language, called G^+, is described in which the database graph is defined as a finite automaton, a query is defined as a regular expression and the output of a query is computed from the intersection of the database graph and a finite automaton representing the query. In particular, the output of a query is the set of ordered pairs, (x, y), such that x and y are nodes in the database graph and there is a *simple path* from x to y (i.e. a path in which no node occurs more than once) labeled by a word which is accepted by the query. (See [CONS89] for a description of a later language, called GraphLog, which is an extension of G^+, and for examples of how GraphLog supports structure-based searching.)

It is shown that query evaluation is, in general, exponential time in the size of the input database graph and that the problem of deciding whether an ordered pair of nodes is in an answer to a given query is NP-complete. Some special cases are exhibited when query answering can be done in polynomial time, in particular query answering is polynomial time when either the database graph is acyclic or the query is a *restricted regular expression* [MEND95]. The contents of nodes are not discussed in [CONS89, MEND95].

In contrast to HQL, the query language G^+ [MEND95] does not utilize temporal logic, which syntactically restricts trail queries to be equivalent to *star-free* regular languages. In addition, the navigation semantics of G^+ [MEND95] process only simple paths in the database graph while the navigation semantics of HQL process trails which may not be simple paths. Finally, we claim that HQL is a more natural formalism for expressing queries in Hypertext than the language of regular expressions, since it is closer in nature to the navigational requirements of Hypertext.

In [STOT89] *Petri nets* [PETE81] are suggested as an underlying formalism for the specification of navigation semantics. Since Petri nets are inherently a concurrency model, they provide a natural semantics for concurrent navigation paths. Furthermore, finite automata are a special case of Petri nets [PETE81] and thus as a special case simplified navigation semantics based on finite automata can be supported.

An important feature of the model presented in [STOT89] is the separation of content from structure, in the sense that the contents of the database are described via a mapping from the nodes of the Petri net (called *places* in Petri net terminology [PETE81]) to the actual pages of information.

Although Petri nets are amenable to formal analysis and constructing the reachability tree for a Petri net is decidable, in general, the complexity of the reachability problem is intractable; in fact, it has been shown that the reachability problem for Petri nets is EXPSPACE-hard [PETE81]; for special cases of Petri-nets when the reachability problem is easier see [ESPA94]. Thus there is a trade-off between expressiveness and complexity, which implies that in practice only special cases of the reachability problem can be incorporated into the navigation semantics of the said model.

In [STOT92] a Hypertext database (called a *hyperdocument*) is viewed as a finite automaton, called the *link automaton*. A *branching temporal logic* [EMER90] language, called HTL*, is proposed for the specification of properties that should be exhibited when navigating through a Hypertext database. HTL* is based on *full branching-time logic* (CTL*) and a subset of HTL*, called HTL, is based on a subset of CTL*, called *computational tree logic* (CTL) [CLAR86, EMER90]. HTL* adds direction to CTL* formulae in order to allow both forward and backward navigation paths to be specified. The propositions of HTL* are atomic predicates, which are conditions to be satisfied at specific states of the link automaton. Correspondingly, the formulae of HTL* are viewed as assertions about the navigation paths of the link automaton. Model checking of an HTL* formula is then used in order to verify that the assertions specified by the formula are satisfied in the link automaton.

Although model checking for a CTL formula can be done in linear time in the length of the formula and the temporal structure being verified, model checking for a CTL* formula was shown to be PSPACE-complete [CLAR86]. In practice, it would be useful to investigate a language whose model checking complexity is in between HTL and HTL*, since HTL may turn out to be too restrictive.

There are several differences between the model presented in [STOT92] and our model. Firstly, we are interested in querying a Hypertext database while Stotts et al. [STOT92] are interested in verifying that the Hypertext database satisfies a set of specifications. In fact, a query can also be viewed as a specification that is satisfied if and only if the set of trails that satisfy the query is not empty. Secondly, branching temporal logic is used in [STOT92] while we use linear time temporal logic. This difference is more fundamental, since in querying a Hypertext database we

are only interested in the set of trails that satisfy a trail query independently of other trails in the database graph. Thirdly, in [STOT92] results from the area of specification and verification of concurrent programs are directly used, while as mentioned in Section 1 there is a fundamental difference between the semantics of PLTL and HQL, in that models of HQL queries are trails which are finite paths, while models of PLTL formulae (and also of path formulae of CTL and CTL* [CLAR86, EMER90]) are infinite paths. Lastly, Stotts et al. [STOT92] do not investigate any specific expressiveness or complexity results relating to their model and concentrate mainly on the definition of their model and showing how it can be applied to other models of Hypertext such as Trellis [STOT92] and Hyperties [RIVL94]. On the other hand, the main aim of our work is actually to investigate the expressiveness and complexity of HQL.

Recently Beeri and Kornatzky [BEER94] have proposed a query language for Hypertext databases which is based on branching temporal logic, with the provision for generalized path quantifiers (which capture natural language assertions) over the trails that satisfy a query. Query answering in their language can be computed in polynomial time in the size of the database graph, since only trails of some fixed bounded length are considered. In HQL no quantifiers are allowed, since its semantics are based on linear time temporal logic. Moreover, we do not bound the lengths of trails under consideration, and thus as we show herein, query answering in our model cannot always be computed in time polynomial in the size of the database graph. Indeed one conclusion of our results may be that for practical purposes the lengths of trails must be bounded, but this comes at the cost of limiting the notion of a trail and thus forcing the user to tune the bounds of the lengths of trails for specific queries.

3   A Formal Model for Hypertext Based on Temporal Logic

In this section we formalize our model for Hypertext which was motivated in the introduction. We begin by introducing the notation.

We denote the cardinality of a set, S, by $|S|$ and the length of a string, w, by $||w||$. In addition, we denote the maximum of two natural numbers m and n by $\max(m, n)$. We will use the O-notation for measuring the computational complexity of algorithms [GARE79]. Finally, we abbreviate "if and only if" to iff.

In the sequel we will be using the following disjoint primitive domains of countably infinite sets:

1. The set of all natural numbers, denoted by ω.

2. The set of all finite length strings, Σ^*, over a finite nonempty alphabet Σ; the empty string is denoted by ϵ, concatenation of two strings w and z will be denoted by wz and a string y is a substring of a string w iff there exist strings x and z such that w = xyz.

3. A set of attribute names (or simply attributes), denoted by $\mathcal{U}$.

4. A set of variables, denoted by $\mathcal{V}$.

Definition 3.1 (Projection) We define the two *projection* operators α and β, such that $\alpha((x, y)) = x$ and $\beta((x, y)) = y$, where (x, y) is an ordered pair of values.

Definition 3.2 (Page) A *page* is an attribute-value pair of the form (A, w), where $A \in \mathcal{U}$ and $w \in \Sigma^*$.

We now give the basic definitions pertaining to arrays [TREN73].

Definition 3.3 (Array) Let N = {1,...,n} be a finite index set, where n $\in \omega$.

An *array* is a family $\{t_i\}$ of *items* (i $\in$ N); each i $\in$ N is called an *index*. (At this stage the definition of the type of items in an array is left unspecified.)

We will use the usual convention whereby A[i] denotes the ith item, A(i), of an array, A, when $1 \leq i \leq n$, otherwise A[i] is taken to be undefined. If A[i] = t, then we say that t $\in$ A.

In the special case when n = 0, the array A is *empty* and $\forall$ i $\in \omega$, A[i] is taken to be undefined; we denote the empty array by [ ].

The *count* of an array A, denoted as #A, is the number of items in A, i.e. #A = n.

Two arrays, A_1 and A_2, are *equal*, denoted as $A_1 = A_2$, if $\#A_1 = \#A_2$ and $\forall$ i $\in$ {1,...,$\#A_1$}, $A_1[i] = A_2[i]$.

An array A_1 is a *suffix* of an array A_2, if $\#A_1 \leq \#A_2$ and $\forall$ i $\in$ {1,...,$\#A_1$}, $A_1[i] = A_2[(\#A_2-\#A_1)+i]$. We let A^i denote the suffix of an array, A, satisfying $\#A^i = \#A-i$, where i $\in$ {0,...,#A}.

An array A_1 is a *prefix* of an array A_2, if $\#A_1 \leq \#A_2$ and $\forall$ i $\in$ {1,...,$\#A_1$}, $A_1[i] = A_2[i]$. We let A_i denote the prefix of an array, A, satisfying $\#A_i = i$, where i $\in$ {0,...,#A}.

The *concatenation* of two arrays, A_1 and A_2, denoted by $A_1 A_2$, is an array A_3 such that $\#A_3 = \#A_1 + \#A_2$, $A_{3\#A_1} = A_1$ and $A_3{}^{\#A_1} = A_2$.

An array, A, is said to be *simple* whenever $\forall$ i $\in$ {1,...,#A}, if i $\neq$ j, then A[i] $\neq$ A[j].

An *information repository* is a simple array of pages, i.e. it satisfies the constraint that pages are unique within the repository. The formal definition follows.

Definition 3.4 (Information repository) An *information repository* (or simply a repository) is a simple array, I, whose items are pages. The indices of I are called the *page numbers* of I. We let ||I|| denote the length of the string I[1]I[2]...I[#I].

The constraint that a repository be a simple array is similar to entity integrity [CODD79] and thus avoids duplication of information. Furthermore, the assumption that a repository is an array, which is *not* nested, is similar to the *first normal form assumption* of tuples in relational databases [ULLM88].

A reachability relation R over a repository, I, is a set of ordered pairs of page numbers of I. The formal definition follows.

Definition 3.5 (Reachability relation) A *reachability relation*, R, over a repository I (or simply a reachability relation if I is understood from context) is a binary relation in {1,...,#I}.

When R corresponds to an acyclic directed graph [BUCK90], we say that R is *acyclic*, otherwise R is *cyclic* (unless explicitly stated otherwise we assume that R is cyclic).

It follows that a reachability relation corresponds to a directed graph. A *trail* in a reachability relation, R, is an array of page numbers corresponding to a path in R. The full definition follows.

Definition 3.6 (Trail) A *trail*, T, in a reachability relation, R, over a repository, I, (or simply a trail in R if I is understood from context) is an array of page numbers of I such that $\forall$ i $\in$ {1,...,#T−1}, (T[i], T[i+1]) $\in$ R. The indices of T are called the *markers* of T.

A *trail* which is a simple array is called a *loop-free* trail. A *subtrail* of a trail, T in R, is a trail in R that is a suffix of a prefix of T (or equivalently a prefix of a suffix of T).

The string induced by a trail T in R, denoted by $\rho(T)$, is defined by $\rho(T) = T[1]T[2]...T[\#T]$.

In general, a trail in a reachability relation, R, corresponds to a path in the directed graph representing R. There are two special cases worth mentioning: a trail of count zero which corresponds to the empty array and a trail of count one which corresponds to a single page number in I.

The following proposition states that the set of trails in R is *suffix closed* and *prefix closed* (cf. [EMER90]).

Proposition 3.1 *The following statements are true:*

1. A trail T is in a reachability relation, R, iff $\forall\ i \in \{0,\dots,\#T\}$, T^i is in R.

2. A trail T is in a reachability relation, R, iff $\forall\ i \in \{0,\dots,\#T\}$, T_i is in R. $\square$

We next define Hypertext databases.

Definition 3.7 (Hypertext database) A *Hypertext database* (or simply a database) H is an ordered pair $<$I, R$>$, where I is a repository and R is a reachability relation over I.

We note that there are no constraints on the reachability relation, R. A Hypertext database $<$I, R$>$ can be viewed as a *temporal structure* [EMER90], where the page numbers of I are associated with the states of the structure and the binary relation R is associated with the transition relation of the structure (we do not assume that R is total as is the case in [EMER90]).

From now on we will assume that $H = \ <$I, R$>$ is a Hypertext database.

4   A Query Language for Navigating in Hypertext

In this section we define the syntax and semantics of the *Hypertext Query Language* (HQL).

We will assume that strings in Σ^* are distinguished by strings beginning with lowercase letters, attribute names in $\mathcal{U}$ are distinguished by strings beginning with uppercase letters excluding X, Y and Z, and variables are distinguished by strings beginning with the uppercase letters X, Y or Z.

Definition 4.1 (Normal and unique variables) We assume that $\mathcal{V}$ is partitioned into two countably infinite sets of variables called *normal variables* and *unique variables*.

Normal variables are distinguished by strings beginning with the uppercase letter X, and unique variables are distinguished by strings beginning with the uppercase letter Y. Variables which may be either normal or unique are distinguished by strings beginning with the uppercase letter Z.

In an interpretation, defined below, both normal and unique variables map to page numbers. However, unique variables restrict the mapping such that no two unique variables can map to the same page number.

Informally, a condition is a propositional logic formula such that its atomic formulae are binary predicates.

Definition 4.2 (Condition) Assume a countably infinite set of binary predicate letters. A *numeric term* is either a natural number or a variable and a *symbolic term* is either an attribute or a string. An *atomic formula* is an expression of the form $\mathrm{P}(t_1, t_2)$, where P is a binary predicate letter, t_1 is a numeric term and t_2 is a symbolic term. An atomic formula, $\mathrm{P}(t_1, t_2)$, is said to be *ground* if t_1 is a natural number, otherwise if t_1 is a variable then $\mathrm{P}(t_1, t_2)$ is said to be *nonground*.

We recursively define the class of *conditions* using the following rules:

C1 A nonground atomic formula $\mathrm{P}(t_1, t_2)$ is a condition.

C2 If C is a condition, then $\neg$(C) is a condition.

C3 If C_1 and C_2 are conditions, then $(C_1 \wedge C_2)$ is a condition.

C4 If C is a condition, then $|\mathbf{V}| = 1$, where $\mathbf{V}$ is the set of variables appearing in C; the single variable appearing in C is called *the variable* of C.

We note that ground atomic formulae are excluded from conditions, since pages are identified by their content and not by their page number. As usual $C_1 \vee C_2$ stands for $\neg(\neg(C_1) \wedge \neg(C_2))$. Also, when no ambiguity arises we omit parentheses in conditions. In addition, the *length* of a condition, C, is the number of symbols in C assuming that no parentheses are omitted.

An interpretation gives meaning to the predicate letters and variables in conditions such that predicate letters are mapped to polynomial time algorithms and variables are mapped to page numbers.

Definition 4.3 (An interpretation) An *interpretation* σ over a Hypertext database H = <I, R> (or simply an interpretation σ if *H* is understood from context) assigns an appropriate meaning to binary predicate letters and terms as follows:

- If P is a binary predicate letter, then $\sigma(P)$ is a mapping from $\omega \times (\mathcal{U} \cup \Sigma^*)$ to {*true, false*} such that $\sigma(P)$ is a polynomial time algorithm [GARE79] in $||I||$.

- If t is either a natural number, an attribute or a string, then $\sigma(t) = t$.

- If Z is a variable, then $\sigma(Z) \in \{1,\dots,\#I\}$, with the constraint that the restriction [HALM74] of σ to unique variables is a one-to-one mapping.

Restricting σ to be a one-to-one mapping, when its domain is the set of unique variables, implies that two distinct unique variables are mapped by σ to distinct page numbers and this allows us to assert the inequality of page numbers.

Definition 4.4 (Satisfaction in an interpretation) Let σ be an interpretation and let C be a condition. Then we say that σ *satisfies* C, written $\sigma \models$ C, provided C is true under the interpretation σ in the usual sense. Specifically,

C1 $\sigma \models P(t_1, t_2)$ iff $\sigma(P)(\sigma(t_1), \sigma(t_2)) = true$.

C2 $\sigma \models \neg(C)$ iff $\sigma \not\models$ C.

C3 $\sigma \models (C_1 \wedge C_2)$ iff $\sigma \models C_1$ and $\sigma \models C_2$.

The following proposition follows by a straightforward induction on the length of a condition.

Proposition 4.1 *Let σ be an interpretation over a Hypertext database H = <I, R> and let C be a condition. Then $\sigma \models C$ can be evaluated in polynomial time in $||I||$ and the length of C.* $\square$

Hereafter the binary predicate letters, *substr* and *att*, will be utilized. For all interpretations σ over <I, R> their meaning is fixed as follows:

1. $\sigma(substr)$ is the substring pattern matching algorithm, that is, $\sigma \models substr(t_1, t_2)$ iff $\sigma(t_2)$ is a substring of $\beta(I[\sigma(t_1)])$.

2. $\sigma(att)$ is the attribute equality algorithm, that is, $\sigma \models att(t_1, t_2)$ iff $\alpha(I[\sigma(t_1)]) = \sigma(t_2)$.

Both $\sigma(substr)$ and $\sigma(att)$ can be evaluated in polynomial time in $||I||$ (see [SEDG90] for efficient substring pattern matching).

We next define the notion of a trail formula and its satisfaction utilizing a *temporal logic* framework [EMER90]. In particular, we will employ propositional linear temporal logic (PLTL) with discrete time.

The temporal operators that will be utilized in the context of a Hypertext database are: $\bigcirc$ which means "nexttime" (one step at a time navigation) and $\diamond$ which means "sometimes" (several

steps at a time navigation). We also define an additional temporal operator, denoted by $\triangle$, which means "finaltime" (reaching the last step of navigation). For simplicity, in this paper, we do not consider the "until" temporal operator, which would add expressive power to our query language [GABB80, EMER90]. From now on, we will refer to these temporal operators as *trail operators*.

Strictly speaking, when we refer to *time* we are actually referring to a *position* in a trail. As pointed out by [RESC71] there is a close connection between *positional* and temporal logic which motivates our use of a subset of PLTL as a query language for Hypertext.

We next define trail formulae, which are similar to PLTL formulae [EMER90]. For simplicity, at this stage, we do not consider negation or disjunction in formulae.

Definition 4.5 (Trail formulae) We recursively define the class of *trail formulae* (or simply formulae) using the following rules:

T1 A condition C is a trail formula.

T2 If f_1 and f_2 are trail formulae, then $(f_1 \wedge f_2)$ is also a trail formula.

T3 If f is a trail formula, then $\Diamond(f)$ is also a trail formula.

T4 If f is a trail formula, then $\bigcirc(f)$ is also a trail formula.

T5 If f is a trail formula, then $\triangle(f)$ is also a trail formula.

When no ambiguity arises we omit parentheses in formulae.

Example 2 We now demonstrate the usefulness of HQL with several example formulae and their intuitive semantics, where we assume that the substring pattern matching algorithm is case insensitive.

1. The formula

$$\Diamond(substr(X_1, local\ news)) \wedge \Diamond(substr(X_2, world\ news)) \wedge$$
$$\Diamond(substr(X_3, sport)) \wedge \Diamond(substr(X_4, weather)),$$

specifies the trails that have a page of local news, a page of world news, a page of sports news and a page with the weather information.

2. The formula

$$substr(X_1, UK\ news) \wedge \bigcirc(substr(X_2, South\ East\ news)) \wedge$$
$$\bigcirc \bigcirc (substr(X_3, London\ news)) \wedge \triangle(substr(X_4, North\ London\ news)),$$

specifies the trails whose first page contains UK news, followed by a page containing South East news, followed by a page containing London news, and a final page containing North London news.

3. The formula

$$\Diamond(substr(X_1, five\ star\ hotels) \wedge substr(X_1, Austin\ Texas)) \wedge$$
$$\bigcirc(substr(X_2, five\ star\ hotels) \wedge substr(X_1, Dallas\ Texas))$$

specifies the trails that have a page of five star hotels in Austin, Texas, followed by a page of five star hotels in Dallas, Texas.

4. The formula

$$substr(X_1, underground) \wedge \neg(substr(X_1, delayed)) \wedge$$
$$\triangle((substr(X_2, buses)) \wedge \neg(substr(X_2, delayed)))$$

specifies the trails whose first page gives us the information about the underground lines that are running normally, and whose last page gives us the information about the bus lines that are running normally.

Definition 4.6 (Subformulae) The set of subformulae of a formula, *f*, is defined recursively as follows:

1. *f* is a subformula of *f*.

2. If *f* is of the form $f_1 \wedge f_2$ then f_1 and f_2 are subformulae of *f*.

3. If *f* is any of the forms $\diamond(f')$, $\bigcirc(f')$ or $\triangle(f')$, then f' is a subformula of *f*.

We denote the set of all subformulae of a trail formula, *f*, by $Sub(f)$. The *length* of a formula, *f*, is the number of symbols in *f* assuming that all conditions have the same length of one and that no parentheses are omitted. We denote the length of *f* by $|f|$. It therefore follows that $|Sub(f)| \leq |f|$.

The reason we have assumed that all conditions *have the same length of one* is that, from a database point of view, the purpose of conditions is to retrieve the set of pages satisfying the condition. Herein we are not interested in the internal structure of conditions, since by Proposition 4.1 we can test whether a page satisfies a condition in polynomial time.

Definition 4.7 (Satisfaction of a trail formula) Let $H = <$I, R$>$ be a Hypertext database, σ be an interpretation over H and T be a trail in R. Then we say that T *satisfies* a formula, *f*, with respect to H and σ (or simply T satisfies *f* if H and σ are understood from context), written $T \models f$, provided *f* is true under T. Specifically,

T1 $T \models C$, iff $\#T > 0$, $\sigma \models C$ and $\sigma(Z) = T[1]$, where Z is the variable of C.

T2 $T \models f_1 \wedge f_2$ iff $T \models f_1$ and $T \models f_2$.

T3 $T \models \diamond(f)$ iff $\exists$ i $\in \{0,\ldots,\#T{-}1\}$ such that $T^i \models$ f.

T4 $T \models \bigcirc(f)$ iff $\#T > 1$ and $T^1 \models$ f.

T5 $T \models \triangle(f)$ iff $\#T > 0$ and $T^i \models f$, where i $= \#T{-}1$.

From Definition 4.7 it follows that for all trail formulae, *f*, $[\ ] \not\models$ f.

Definition 4.8 (A model of a trail formula) A Hypertext database $H = <$I, R$>$ is a *model* of a trail formula, *f*, if $\exists$ T in R such that $T \models f$ with respect to H and some interpretation σ. Whenever H is understood from context we also say that T is a model of *f*.

Definition 4.9 (Trail equivalence) A formula f_1 *trail implies* (or simply implies) a formula f_2, written $f_1 \Rightarrow f_2$, if

for all Hypertext databases $< I, R >$, for all trails T in R, if $T \models f_1$ then $T \models f_2$.

A formula f_1 is *trail equivalent* (or simply equivalent) to a formula f_2, written $f_1 \equiv f_2$ if $f_1 \Rightarrow f_2$ and $f_2 \Rightarrow f_1$.

The following proposition gives the significant equivalences and implications between HQL formulae. We note that the implications given in the proposition cannot be strengthened to equivalences.

Proposition 4.2 *The following equivalences and implications are satisfied:*

1. $\Diamond(f) \equiv \Diamond\Diamond(f)$.

2. $\triangle(f) \equiv \triangle\triangle(f)$.

3. $\Diamond \bigcirc (f) \equiv \bigcirc\Diamond(f)$.

4. $\triangle(f) \equiv \triangle\Diamond(f) \equiv \Diamond\triangle(f)$.

5. $\bigcirc(f_1 \wedge f_2) \equiv \bigcirc(f_1) \wedge \bigcirc(f_2)$.

6. $\triangle(f_1 \wedge f_2) \equiv \triangle(f_1) \wedge \triangle(f_2)$.

7. $\Diamond(f_1 \wedge f_2) \Rightarrow \Diamond(f_1) \wedge \Diamond(f_2)$.

8. $f \Rightarrow \Diamond(f)$.

9. $\bigcirc(f) \Rightarrow \Diamond(f)$.

10. $\triangle(f) \Rightarrow \Diamond(f)$.

11. $\bigcirc\triangle(f) \Rightarrow \triangle(f)$. $\square$

The following definition will be useful when we consider subclasses of HQL.

Definition 4.10 (Op-free trail formulae) A trail formula is *op-free* if it does not contain any occurrences of *op*, where $op \in \{\Diamond, \bigcirc, \triangle\}$.

The $\Diamond$ operator is declarative since it does not specify the precise navigation sequence, while the $\bigcirc$ operator is procedural since it progresses navigation one step at a time. Thus $\bigcirc$-free trail queries can be viewed as declarative queries and $\Diamond$-free trail queries can be viewed as procedural queries. On the other hand, $\triangle$-free trail queries are queries which do not specify any condition on the final item of a trail which satisfies the query.

The next proposition shows that the general class of trail formulae is more expressive than the class of $\triangle$-free trail formulae. That is, if we remove $\triangle$ from HQL we lose expressive power.

Proposition 4.3 *It is not the case that for all $\triangle$-free trail formulae, g, there exists a $\triangle$-free trail formula, f, such that $f \Rightarrow \triangle(g)$.*

Proof. The result follows by a straightforward induction on the length of *f*.

(*Basis*): If $|f| = 1$, then *f* is a condition, say C. Without loss of generality, let $H = <I, R>$ be a Hypertext database having a trail T in R with $\#T = 2$ and such that $T \models C$ but $T^1 \not\models g$. The result that $T \not\models \triangle(g)$ follows by Definition 4.7 part (T5).

(*Induction*): Assume that the result holds when $|f| = k$, where $k \geq 1$; we then need to prove that the result holds when $|f| > k$.

We now consider in turn the different cases pertaining to the structure of f:

1. If *f* is the trail formula $f_1 \wedge f_2$, then by the inductive hypothesis there exists a Hypertext database $H = <I, R>$ and a trail T in R such that $T \models f_1$ and $T \models f_2$ but $T \not\models \triangle(g)$. The result follows by Definition 4.7 part (T2), since $T \models f$.

2. If f is the trail formula $\Diamond(f')$, then by the inductive hypothesis there exists a Hypertext database $H = <I, R>$ and a trail T in R such that $T^i \models f'$ but $T^i \not\models \triangle(g)$, where i $\in$ $\{0,\ldots,\#T-1\}$. The result that T $\models f$ but T $\not\models \triangle(g)$ follows by Definition 4.7 part (T3) and part (T5), respectively.

3. If f is the trail formula $\bigcirc(f')$, then by the inductive hypothesis there exists a Hypertext database $H = <I, R>$ and a trail T in R such that $\#T > 1$ and $T^1 \models f'$ but $T^1 \not\models \triangle(g)$. The result that T $\models f$ but T $\not\models \triangle(g)$ follows by Definition 4.7 part (T4) and part (T5), respectively. $\square$

A *trail query* is a trail formula viewed as a mapping from Hypertext databases to sets of trails, where a trail is in the output of the trail query iff it satisfies the trail formula with respect to the input Hypertext database and some interpretation. The formal definition follows.

Definition 4.11 (Trail query) A *trail query* (or simply an HQL query or a query) is a trail formula, f, viewed as a mapping from Hypertext databases to sets of trails such that, given an input Hypertext database $H = <I, R>$, the output *f(H)* is defined by

$$\{T \mid T \text{ is a trail in R and } T \models f\}.$$

A trail T in R satisfying T $\models f$ is called *an answer* of *f(H)*.

5 Finite Automata and Trail Queries

In this section we utilize the theory of finite automata [HOPC79, PERR90] in order to construct the output of a query, which in the general case may consist of a countably infinite set of trails. In particular, we exploit the strong connection between finite automata accepting star-free regular languages and PLTL [VARD86a, VARD86b, EMER90, THOM90].

Definition 5.1 (Hypertext automaton) The *Hypertext automaton* representing a Hypertext database, $H = <I, R>$, (or simply the Hypertext automaton whenever H is understood from context) is a finite automaton defined by a quintuple of the form $\mathcal{M}_H = (\mathcal{A}, Q, \Delta, S, F)$ (or simply $\mathcal{M}$ whenever H is understood from context), where

- $\mathcal{A} = \{1,\ldots,m\}$ is a finite alphabet with m = #I.

- $Q = \{s_1, \ldots, s_m, s_{m+1}, \ldots, s_{2m}\}$ is a set of 2m states.

- $\Delta \subseteq Q \times \mathcal{A} \times Q$ is a transition relation, where $(s_j, \delta(s_j), s_k) \in \Delta$ iff j $\in \{1,\ldots,m\}$ and either k $\in \{1,\ldots,m\}$, with $(\delta(s_j), \delta(s_k)) \in R$, or k = m + j, where δ is a one-to-one and onto mapping from $\{s_1, \ldots, s_m\}$ to $\mathcal{A}$ such that $\delta(s_i) = i$, $1 \le i \le m$.

- $S = \{s_1, \ldots, s_m\}$ is the set of initial states.

- $F = \{s_{m+1}, \ldots, s_{2m}\}$ is the set of terminal states.

If S $= \emptyset$ and thus F $= \emptyset$, then $\mathcal{M}$ is called the *empty Hypertext automaton*.

We observe that the construction of $\mathcal{M}_H$ is independent of the actual contents of the pages in the repository I of *H*. It only depends on the cardinality, #I of I, and the reachability relation R of *H*. In other words, a finite automaton representing a Hypertext database is independent of the actual data stored in the repository and thus induces an equivalence class of Hypertext databases having repositories of the same cardinality and having the same reachability relation.
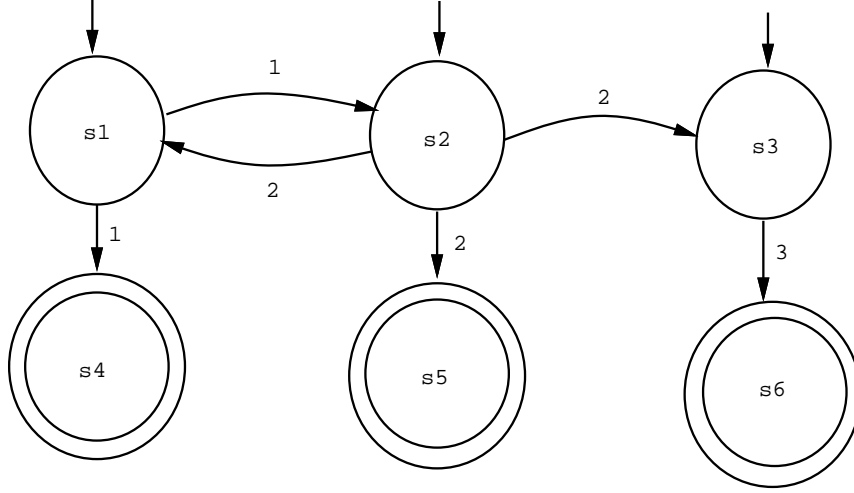
Figure 2: A finite automaton for a Hypertext database

Example 3 The finite automaton $\mathcal{M}_H$ is shown in Figure 2, where $H = <$I, R$>$, $\#$I $= 3$ and R $= \{(1,2), (2,1), (2,3)\}$.

Definition 5.2 (The language accepted by a finite automaton) A *path* in the finite automaton, $\mathcal{M} = (\mathcal{A}$, Q, Δ, S, F) (or simply a path if $\mathcal{M}$ is understood from context), is a nonempty sequence $\{(s_i, \delta(s_i), s_{i+1})\}$ ($i \in$ N) of transitions in Δ, with N $= \{1,\ldots,n\}$ and n $\in \omega$. (We also say that there exists a path from $(s_1, \delta(s_1), s_2)$ to $(s_n, \delta(s_n), s_{n+1})$.) If $s_1 = s_{n+1}$, then the path is a *cycle*. The string w $= \delta(s_1)\delta(s_2)\ldots\delta(s_n)$ is called the *label* of the path, s_1 is called its *origin* and s_{n+1} is called its *end*. The *length* of the path is n.

A path is *successful* if its origin is in S and its end is in F. A string w $\in \mathcal{A}^*$ is said to be *accepted* by $\mathcal{M}$ if it is the label of some successful path. The language accepted by $\mathcal{M}$, denoted by $\mathcal{L}(\mathcal{M})$, is the set of all strings accepted by $\mathcal{M}$.

When $\mathcal{M}$ is the empty Hypertext automaton, $\mathcal{L}(\mathcal{M}) = \emptyset$, that is, the empty language is accepted. Furthermore, there does not exist a Hypertext database H such that $\mathcal{L}(\mathcal{M}_H) = \{\epsilon\}$, i.e. no finite automaton representing a Hypertext database accepts the empty string. Thus only subsets of $\mathcal{A}^+ = \mathcal{A}^* - \{\epsilon\}$ are accepted by finite automata representing Hypertext databases.

We proceed to show that $\mathcal{L}(\mathcal{M}_H)$ is a *star-free* regular language [MCNA71, PERR90] (or simply a star-free language).

Definition 5.3 (Star-free languages) A regular language is *star-free* if it can be generated from a finite set of strings by repeated applications of the Boolean operations, union, intersection and complementation (with respect to $\mathcal{A}^*$), together with concatenation.

The following definition is needed in order to state an alternative characterization of star-free languages.

Definition 5.4 (Aperiodic regular languages) A regular language $\mathcal{L} \subseteq \mathcal{A}^*$ is *aperiodic* if $\exists$ n $\in \omega$ (n > 0) such that $\forall$ x,y,z $\in \mathcal{A}^*$,

$$xy^n z \in \mathcal{L} \text{ iff } xy^{n+1}z \in \mathcal{L}, \tag{1}$$

where y^n is the concatenation of y with itself n times.

The following theorem states the equivalence of star-free and aperiodic regular languages [PERR90, Theorem 6.1].

Theorem 5.1 *A regular language is* star-free *iff it is aperiodic.*   □

Theorem 5.2 $\mathcal{L}(\mathcal{M}_H)$ *is a star-free language.*

Proof. Let $\mathcal{L} = \mathcal{L}(\mathcal{M}_H)$. In order to prove the result we use Theorem 5.1 to show that (1) is satisfied, with n = 2. If $y = \epsilon$, then the result holds trivially, so we assume that this is not the case. Next assume that $xy^2z \in \mathcal{L}$ and let $y = a_1 \ldots a_k = \delta(s_1) \ldots \delta(s_k)$ be the label of the path $\{(s_i, \delta(s_i), s_{i+1})\}$ ($i \in \{1,\ldots,k\}$) in $\mathcal{M}_H$.

It follows that s_i ($i \in \{1,\ldots,k\}$) is one of the initial #I states in Q, i.e. $1 \leq i \leq$ #I, since by Definition 5.1 terminal states in $\mathcal{M}_H$ cannot be the first component of any transition. Furthermore, $s_{k+1} = s_1$, since $a_1\ldots a_k a_1$ is a substring of y^2 and by Definition 5.1 δ is a one-to-one and onto mapping from the initial #I states in Q to $\mathcal{A}$.

Thus, the substring $a_1\ldots a_k$ corresponds to a *cycle* [BUCK90] in the directed graph corresponding to R (where we allow a node to appear more than once in a cycle). The result follows, since it is implied that $xy^2z \in \mathcal{L}$ iff $xy^nz \in \mathcal{L}$, with n $\geq$ 2.   □

It is easy to demonstrate that the converse of Theorem 5.2 does not hold. For example, let $\mathcal{A}$ = {a} and $\mathcal{L}$ = {aa}. $\mathcal{L}$ is obviously star-free, since it is a finite regular language, but there does not exist a Hypertext database H such that $\mathcal{L} = \mathcal{L}(\mathcal{M}_H)$. Thus, the class of languages accepted by finite automata representing Hypertext databases is a proper subclass of the class of star-free languages.

Definition 5.5 (The language induced by a trail query) The language induced by a set **T** (possibly countably infinite) of trails in R, denoted by $\mathcal{L}(\mathbf{T})$, is defined by $\mathcal{L}(\mathbf{T}) = \{\rho(\text{T}) \mid \text{T} \in \mathbf{T}\}$. The language induced by the output $f(H)$ of a trail query f is defined to be $\mathcal{L}(f(H))$.

It follows that, if a string w $= \delta(s_1)\delta(s_2)\ldots\delta(s_n)$ is accepted by $\mathcal{M}$, then there exists a trail, T, in R such that $\rho(\text{T})$ = w.

The following lemma states that the language induced by the output of a trail query over a Hypertext database H is a subset of the language accepted by the finite automaton representing H.

Lemma 5.3 $\mathcal{L}(f(H)) \subseteq \mathcal{L}(\mathcal{M}_H)$, *where f is a trail query and* H *is a Hypertext database.*

Proof. The result follows immediately from Definitions 5.1 and 5.5 on using Definition 4.11.   □

The following proposition shows that the finite automaton representing a Hypertext database can be viewed as the output of a certain trail query.

Proposition 5.4 *For all Hypertext databases, $H = <I, R>$, there exists a trail query f such that* $\mathcal{L}(f(H)) = \mathcal{L}(\mathcal{M}_H)$.

Proof. Let S $= \{s_1,\ldots,s_m\}$ be the set of initial states of $\mathcal{M}_H$ and F $= \{s_{m+1},\ldots,s_{2m}\}$ be the set of terminal states of $\mathcal{M}_H$. Furthermore, let pages(S) = pages(F) = {I[i] | $s_i \in$ S} be the set of pages in I corresponding to S and F, respectively. Hereafter we assume that pages(S) = pages(F) $= \{(A_1, w_1),\ldots,(A_m, w_m)\}$.
Let f^S be the trail formula

$$(att(X_1, A_1) \wedge substr(X_1, w_1)) \vee \ldots \vee (att(X_1, A_m) \wedge substr(X_1, w_m))$$

and let f^F be the trail formula

$$\triangle((att(X_2, A_1) \wedge substr(X_2, w_1)) \vee \ldots \vee (att(X_2, A_m) \wedge substr(X_2, w_m))),$$

15

where X_1 and X_2 are distinct normal variables.

Finally, let T be a trail in R and let f be the trail formula $f^S \wedge f^F$. By Definition 4.7 it follows that T $\models f$ iff T $\models f^S$ and T $\models f^F$. Furthermore, T $\models f^S$ iff T[1] $\in$ pages(S) and T $\models f^F$ iff T[#T] $\in$ pages(F). Thus T is an answer of $f(H)$ iff T[1] $\in$ pages(S) and T[#T] $\in$ pages(F). Therefore, $\mathcal{L}(f(H)) = \mathcal{L}(\mathcal{M}_H)$ holds as required. □

We proceed to investigate the correspondence between finite automata and trail queries.

Recall that star-free regular languages are closed under union, intersection and concatenation [HOPC79, PERR90]. We will assume that when taking the union, intersection or concatenation of two finite automata *their sets of states are disjoint*. Furthermore, for convenience we will also assume that the concatenation of any finite automaton with the empty Hypertext automaton yields the empty Hypertext automaton.

We next define a useful finite automaton with respect to a repository, I, which accepts the star-free regular language $\mathcal{A}^+$; we call this finite automaton the *complete automaton*.

Definition 5.6 (The complete automaton of a repository) The complete automaton with respect to a repository, I, is a quintuple of the form $\mathcal{M}_{(\gamma,I)} = (\mathcal{A}, Q_\gamma, \Delta_\gamma, S_\gamma, F_\gamma)$ (or simply $\mathcal{M}_\gamma$ whenever I is understood from context), where

- $\mathcal{A} = \{1,...,m\}$ is a finite alphabet with m = #I.

- $Q_\gamma = \{s\}$ is the singleton set of states.

- $\Delta_\gamma = Q_\gamma \times \mathcal{A} \times Q_\gamma$ is the transition relation.

- $S_\gamma = \{s\}$ is the initial state.

- $F_\gamma = \{s\}$ is the terminal state.

The finite automaton representing a trail query f, with respect to a repository I, (or simply the finite automaton representing f if I is understood from context) is a quintuple of the form $\mathcal{M}_{(f,I)} = (\mathcal{A}, Q_f, \Delta_f, S_f, F_f)$ (or simply $\mathcal{M}_f$ whenever I is understood from context), where as before $\mathcal{A} = \{1,...,m\}$ with m = #I.

Prior to constructing $\mathcal{M}_f$ we present an algorithm, which recursively constructs an intermediate finite automaton, $\mathcal{M}_{f'}$, in accordance with the subformulae of f.

We will assume that with each transition, (s_i, p, s_j), in the transition relation of a finite automaton, say $\mathcal{M}$, we maintain an auxiliary set, denoted by cond($\mathcal{M}, (s_i, p, s_j)$) (or simply cond($s_i, p, s_j$) if $\mathcal{M}$ is understood from context), which is initialized to the empty set. The set cond(s_i, p, s_j) will store the conditions associated with the said transition.

Definition 5.7 (The intermediate finite automaton representing a trail query) As an intermediate step we present an algorithm, which recursively constructs a finite automaton $\mathcal{M}_{f'} = (\mathcal{A}, Q_{f'}, \Delta_{f'}, S_{f'}, F_{f'})$ with f and I given as its inputs; the algorithm is designated by τ^I when I is its input repository (or simply τ when I is understood from context). For the purpose of the algorithm we will assume that τ has an additional implicit Boolean parameter, designated by $\triangle$flag, which is initialized to *false*.

The algorithm considers all of the subformulae $g \in Sub(f)$ in decreasing order of $|g|$ in accordance with the structure of g (recall that we have assumed that all conditions have the same length of one):

1. If g is just the condition C, with variable Z, then let $\{i_1, \ldots, i_k\}$ be the largest subset of $\{1,...,\#I\}$ such that there exists an interpretation σ satisfying $\sigma(Z) = i_j$, $1 \leq j \leq k$, and $\sigma \models C$.

   If k $\geq$ 1, then $\mathcal{M}_C = (\mathcal{A}, Q_C, \Delta_C, S_C, F_C)$, where

- $Q_C = \{s_{C_1}, s_{C_2}\}$.
- $\Delta_C = \{(s_{C_1}, i_1, s_{C_2}), \ldots, (s_{C_1}, i_k, s_{C_2})\}$ and $\forall\, j \in \{1,\ldots,k\}$, set $\mathrm{cond}(s_{C_1}, i_j, s_{C_2})$ to C.
- $S_C = \{s_{C_1}\}$ is the initial state.
- $F_C = \{s_{C_2}\}$ is the terminal state.

If $\triangle$flag is *true*, then $\tau(g)$ returns $\mathcal{M}_C$, otherwise (when $\triangle$flag is *false*) $\tau(g)$ returns the concatenation of $\mathcal{M}_C$ and $\mathcal{M}_\gamma$. Finally, if the aforesaid subset is empty, i.e. k = 0, then $\tau(g)$ returns the empty Hypertext automaton.

2. If g is the formula $g_1 \wedge g_2$, such that $\tau(g_1)$ returns $\mathcal{M}_{g_1}$ and $\tau(g_2)$ returns $\mathcal{M}_{g_2}$, then $\tau(g)$ returns the intersection of $\mathcal{M}_{g_1}$ and $\mathcal{M}_{g_2}$. In addition, if $\mathrm{cond}((s_{i_1}, s_{i_2}), p, (s_{j_1}, s_{j_2}))$ is in the transition relation of the intersection of $\mathcal{M}_{g_1}$ and $\mathcal{M}_{g_2}$, then $\mathrm{cond}((s_{i_1}, s_{i_2}), p, (s_{j_1}, s_{j_2}))$ is set to $\mathrm{cond}(\mathcal{M}_{g_1}, (s_{i_1}, p, s_{j_1})) \cup \mathrm{cond}(\mathcal{M}_{g_2}, (s_{i_2}, p, s_{j_2}))$.

3. If g is the formula $\Diamond(g')$ and $\tau(g')$ returns $\mathcal{M}_{g'}$, then $\tau(g)$ returns $\mathcal{M}_{g'}$ if $\triangle$flag is *true*, otherwise (when $\triangle$flag is *false*) $\tau(g)$ returns the concatenation of $\mathcal{M}_\gamma$ and $\mathcal{M}_{g'}$.

4. If g is the formula $\bigcirc(g')$ and $\tau(g')$ returns $\mathcal{M}_{g'}$, then $\tau(g)$ returns the empty Hypertext automaton if $\triangle$flag is *true*, otherwise (when $\triangle$flag is *false*) $\tau(g)$ returns the concatenation of $\mathcal{M}_\bigcirc$ and $\mathcal{M}_{g'}$, where $\mathcal{M}_\bigcirc = (\mathcal{A}, Q_\bigcirc, \Delta_\bigcirc, S_\bigcirc, F_\bigcirc)$ is defined as follows:

- $Q_\bigcirc = \{s_{\bigcirc_1}, s_{\bigcirc_2}\}$.
- $\Delta_\bigcirc = \{(s_{\bigcirc_1}, 1, s_{\bigcirc_2}), \ldots, (s_{\bigcirc_1}, m, s_{\bigcirc_2})\}$, where $m = \#\mathrm{I}$.
- $S_\bigcirc = \{s_{\bigcirc_1}\}$ is the initial state.
- $F_\bigcirc = \{s_{\bigcirc_2}\}$ is the terminal state.

5. If g is the formula $\triangle(g')$, $\triangle$flag is set to *true*, and if $\tau(g')$ returns $\mathcal{M}_{g'}$, then $\tau(g)$ returns $\mathcal{M}_{g'}$ provided $\triangle$flag was *true* prior to invoking $\tau(g')$, otherwise (when $\triangle$flag was *false* prior to invoking $\tau(g')$) $\tau(g)$ returns the concatenation of $\mathcal{M}_\gamma$ and $\mathcal{M}_{g'}$.

Example 4 Let f be the trail query $\bigcirc\triangle(\mathrm{C})$, with C being a condition such that the only interpretation σ over H, with $\sigma \models \mathrm{C}$, satisfies $\sigma(\mathrm{Z}) = 2$, where Z is the variable of C and 2 is a page number of I. In addition, let I be the information repository of the Hypertext database H of Example 3. The finite automaton for $\mathcal{M}_{f'}$ is shown in Figure 3.

It can be verified that the regular language accepted by the finite automaton shown in Figure 3 would remain the same were f to be the trail formula $\bigcirc\Diamond\triangle(\mathrm{C})$. This can also be deduced from Proposition 4.2 part (4).

We next prove that the language accepted by $\tau(f)$ is a star-free regular language.

Lemma 5.5 $\mathcal{L}(\tau(f))$ *is a star-free regular language.*

Proof. The result follows by a straightforward induction on the length of f, observing that $\mathcal{L}(\mathcal{M}_\gamma) = \mathcal{A}^+$ is star-free, and by the fact that, due to the construction of $\mathcal{M}_g$ via τ, $\mathcal{L}(\mathcal{M}_g)$ is also star-free, for all subformulae g of f. (Recall that a finite regular language is star-free and that a finite automaton having no cycles accepts a finite regular language.) $\square$

The following lemma gives an upper bound on the cardinality of the set of states of the finite automaton $\tau(f)$ constructed by the algorithm given in Definition 5.7. Note that this upper bound is independent of the repository, I, and is exponential only in the number of conjunctions, between subformulae of f, plus one.
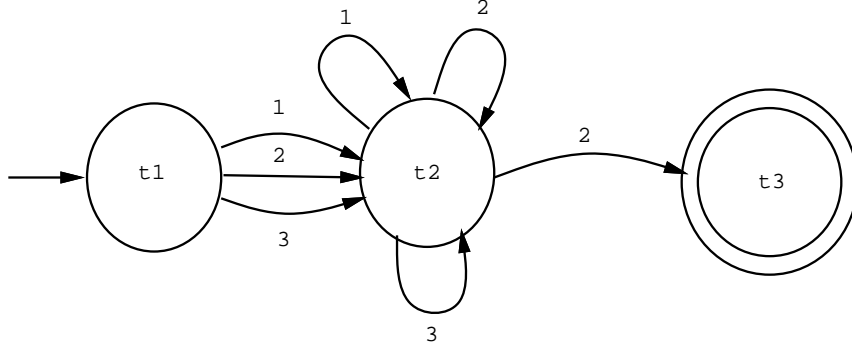
Figure 3: A finite automaton for a trail query

Lemma 5.6 $|Q_{f'}| \leq O(|f|)^{\wedge(f)}$, where f is a trail formula, $\tau(f) = \mathcal{M}_{f'} = (\mathcal{A}, Q_{f'}, \Delta_{f'}, S_{f'}, F_{f'})$ and $\wedge(f)$ is the number of conjunctions, between subformulae of f, plus one.

Proof. We prove the result by induction on $\wedge(f)$. We observe that given two finite automata, whose state sets are Q_1 and Q_2, when we concatenate, or correspondingly, intersect the two finite automata, then the cardinality of the state set of the resulting finite automaton is $|Q_1| + |Q_2|$, or correspondingly, $|Q_1| \cdot |Q_2|$ [HOPC79, PERR90].

Recall that we have assumed that all conditions in f have the same length of one.

(*Basis*): If $\wedge(f) = 1$, then f has no conjunctions. The result that $|Q_{f'}| \leq O(|f|)$ follows by a straightforward induction on the length of f by inspecting Definition 5.7 in accordance with the structure of *f*.

(*Induction*): Assume that the result holds when $\wedge(f) = $ k, where k ≥ 1; we then need to prove that the result holds when $\wedge(f) = $ k+1.

It follows that f must be of the form $op_1(\ldots op_q(g) \ldots)$, where $op = op_1 \ldots op_q$ is a sequence of trail operators, with $q \geq 0$, and g is a subformula of f of the form $g_1 \wedge g_2$, satisfying $\wedge(g_1) \leq $ k and $\wedge(g_2) \leq $ k.

By part (2) of the description of τ in Definition 5.7 $|Q_g| \leq |Q_{g_1}| \cdot |Q_{g_2}|$, where Q_{g_1} and Q_{g_1} are the state sets of the finite automata $\mathcal{M}_{g_1}$ and $\mathcal{M}_{g_2}$, respectively. Furthermore, by the inductive hypothesis $|Q_{g_1}| \leq O(|g_1|)^{\wedge(g_1)}$ and $|Q_{g_2}| \leq O(|g_2|)^{\wedge(g_2)}$. It therefore follows that $|Q_g| \leq O(|f|)^{\wedge(f)}$, since $|g_1| + |g_2| < |f|$ and $\wedge(g_1) + \wedge(g_2) = \wedge(f)$.

The result now follows by a straightforward induction on the number, q, of trail operators in *op* by inspecting Definition 5.7 according to the three different cases where the first operator, op_1, is $\Diamond$, $\bigcirc$ or $\triangle$. $\square$

In order to conclude the construction of $\mathcal{M}_f$ we need to take into account the semantics of normal and unique variables. To accomplish this we use the auxiliary sets $\text{cond}(s_i, p, s_j)$ and modify the output of $\tau(f)$ so that it satisfies the constraints enumerated below.

Firstly, we define some convenient terminology. A *time unit* of a trail formula f (or simply a time unit whenever f is understood from context) is defined to be a transition, (s_i, p, s_j), in the transition relation of the finite automaton $\tau(f)$, where p is a page number of I and s_i, s_j are states. The variable of a condition, C $\in \text{cond}(s_i, p, s_j)$, is said to be *appearing* at the time unit (s_i, p, s_j).

The following two constraints must be enforced on variables appearing at time units:

1. No two unique variables, which are distinct, can appear at two (not necessarily distinct) time units having the same page number (recall Definition 4.1 and the comment that follows).

2. If the same variable appears at two distinct time units, then its corresponding interpretations must be the same.

In order to formalize the above constraints, let $\varphi(\text{cond}(s_i, p, s_j))$ denote the set of variables of the conditions contained in $\text{cond}(s_i, p, s_j)$, where (s_i, p, s_j) is a time unit of a trail formula f. Thus the following conditions, corresponding to the above two constraints, must be enforced in the finite automaton $\tau(f)$.

V1 For all (not necessarily distinct) time units (s_{i_1}, p, s_{j_1}) and (s_{i_2}, p, s_{j_2}), if there exists a path from (s_{i_1}, p, s_{j_1}) to (s_{i_2}, p, s_{j_2}), then there do not exist two unique variables, which are distinct, included in $\varphi(\text{cond}(s_{i_1}, p, s_{j_1})) \cup \varphi(\text{cond}(s_{i_2}, p, s_{j_2}))$.

V2 For all distinct time units, (s_{i_1}, p_1, s_{j_1}) and (s_{i_2}, p_2, s_{j_2}), if there exists a path from (s_{i_1}, p_1, s_{j_1}) to (s_{i_2}, p_2, s_{j_2}) and $\varphi(\text{cond}(s_{i_1}, p_1, s_{j_1})) \cap \varphi(\text{cond}(s_{i_2}, p_2, s_{j_2})) \neq \emptyset$, then $p_1 = p_2$.

We now present a further algorithm, designated by π, that takes as its input the finite automaton $\tau(f)$ and transforms it into the desired finite automaton $\mathcal{M}_f$.

Algorithm 1 ($\pi(f)$)
1. **begin**
2. $\quad$ $\mathcal{M}_f = (\mathcal{A}, Q_f, \Delta_f, S_f, F_f) := \tau(f)$;
3. $\quad$ **while** $\exists (s_{i_1}, p_1, s_{j_1}), (s_{i_2}, p_2, s_{j_2}) \in \Delta_f$ that violate either (V1) or (V2) **do**
4. $\quad\quad$ $\Delta_1 := \Delta_f - \{(s_{i_1}, p_1, s_{j_1})\}$;
5. $\quad\quad$ $\mathcal{M}_1 := (\mathcal{A}, Q_f, \Delta_1, S_f, F_f)$;
6. $\quad\quad$ $\Delta_2 := \Delta_f - \{(s_{i_2}, p_2, s_{j_2})\}$;
7. $\quad\quad$ $\mathcal{M}_2 := (\mathcal{A}, Q_f, \Delta_2, S_f, F_f)$;
8. $\quad\quad$ $\mathcal{M}_f :=$ the union of $\mathcal{M}_1$ and $\mathcal{M}_2$;
9. $\quad$ **end while**
10. $\quad$ **return** $\mathcal{M}_f$;
11. **end.**

Example 5 It can be verified that for the trail query f of Example 4 $\mathcal{M}_{f'} = \mathcal{M}_f$, since both the constraints (V1) and (V2) are satisfied.

In order to prove that the regular language, accepted by the finite automaton, $\mathcal{M}_f = \pi(f)$, output from Algorithm 1, is a star-free regular language we first prove the following lemma.

Lemma 5.7 *If $\mathcal{L}(\mathcal{M})$ is a star-free regular language, then $\mathcal{L}(\mathcal{M}')$ is also a star-free regular language, where $\mathcal{M} = (\mathcal{A}, Q, \Delta, S, F)$ and $\mathcal{M}' = (\mathcal{A}, Q, \Delta - \{(s_i, p, s_j)\}, S, F)$ are finite automata.*

Proof. Let $\mathcal{L} = \mathcal{L}(\mathcal{M})$ and $\mathcal{L}' = \mathcal{L}(\mathcal{M}')$. Also, assume that $(s_i, p, s_j) \in \Delta$, otherwise the result follows trivially. We note that by the construction of $\mathcal{M}'$ we have $\mathcal{L}' \subseteq \mathcal{L}$.

We say that an occurrence of an alphabet symbol p in a substring y of w $\in \mathcal{A}^*$ is *necessary* with respect to (s_i, p, s_j) (or simply p $\in$ y is necessary if w and (s_i, p, s_j) are understood from context) whenever the following statement is satisfied:

If w is accepted by $\mathcal{M}$ and p is the ith symbol in w, then (s_i, p, s_j) is the ith transition of *all* successful paths whose label is w.

Since $\mathcal{L}$ is a star-free regular language, then from Theorem 5.1 $\exists$ n $\in \omega$ (n > 0) such that $\forall$ x,y,z $\in \mathcal{A}^*$, (1) is satisfied; let us fix n to be such a natural number.

In order to conclude the proof we show that $\exists$ m (m $\geq$ n) such that $\forall$ x,y,z $\in \mathcal{A}^*$,

$$xy^m z \in \mathcal{L}' \text{ iff } xy^{m+1} z \in \mathcal{L}'.$$

Let us fix m = n+1 and x, y, z $\in \mathcal{A}^*$ to be arbitrary strings.

By Theorem 5.1 $xy^m z \in \mathcal{L}$ iff $xy^{m+1} z \in \mathcal{L}$. Furthermore, if $xy^m z \in \mathcal{L}$, then it is also the case that $xy^{m-1} z \in \mathcal{L}$, since n = m−1, and thus the mth occurrence of y in y^m must be the label of a cycle that causes the string $xy^m z$ to be in $\mathcal{L}$.

It follows that the following two statements are equivalent:

1. $xy^m z \in \mathcal{L}'$ iff $xy^{m+1}z \in \mathcal{L}'$.

2. The number of necessary occurrences of p in the m*th* occurrence of the substring y of $xy^m z \in \mathcal{L}$ is zero iff the number of necessary occurrences of p in the (m+1)*th* occurrence of the substring y of $xy^{m+1}z \in \mathcal{L}$ is also zero.

It remains to show that statement (2) above is true.

The *if part* of statement (2) implies that there exists a cycle, say θ, whose label is y, that causes $xy^{m+1}z$ to be in $\mathcal{L}$ and does not include (s_i, p, s_j). Therefore, θ also causes $xy^m z$ to be in $\mathcal{L}$, since n = m−1, implying that there exists a cycle, whose label is y, that causes $xy^m z$ to be in $\mathcal{L}$. It follows that the number of necessary occurrences of p in the m*th* occurrence of the substring y of $xy^m z \in \mathcal{L}$ is zero, as required.

Similarly, the *only if part* of statement (2) implies that there exists a cycle, whose label is y, that causes $xy^m z$ to be in $\mathcal{L}$ and does not include (s_i, p, s_j). Therefore, the number of necessary occurrences of p in the (m+1)*th* occurrence of the substring y of $xy^{m+1}z \in \mathcal{L}$ is also zero, as required. $\square$

We proceed to show that $\mathcal{L}(\mathcal{M}_f)$ is a star-free regular language. In particular, the class of languages accepted by finite automata representing trail queries is a proper subclass of the class of star-free regular languages. (Recall that trail formulae do not include the until temporal operator and that negation and disjunction are included only in conditions; see [EMER90, Theorem 6.4].)

Theorem 5.8 $\mathcal{L}(\mathcal{M}_f)$ *is a star-free regular language.*

Proof. By Lemma 5.5 the regular language $\mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'} = \tau(f)$, is star-free. In addition, by Lemma 5.7 the regular languages accepted by the finite automata, $\mathcal{M}_1$ and $\mathcal{M}_2$, of Algorithm 1 are both star-free. Furthermore, the regular language accepted by $\mathcal{M}_f$, the finite automaton assigned a value at line 8 of the said algorithm, is also star-free, since star-free regular languages are closed under union. Thus, the transformation carried out via the while loop of Algorithm 1, beginning at line 3 and ending at line 9, preserves star-freeness of the regular language accepted by $\mathcal{M}_f$. The result now follows. $\square$

The next theorem follows by a straightforward inspection of Algorithm 1 noting that given the auxiliary set, cond(s_i, p, s_j), $\sum_q |C_q| \leq |f|$, where $C_q \in$ cond(s_i, p, s_j). (See Algorithm 2 in Section 6, whose polynomial time complexity is $O(|Q_f| \cdot |\Delta_f|)$, which can be used to decide whether there exists a path from (s_{i_1}, p_1, s_{j_1}) to (s_{i_2}, p_2, s_{j_2}), where $\pi(f) = \mathcal{M}_f = (\mathcal{A}, Q_f, \Delta_f, S_f, F_f)$.)

Theorem 5.9 *The finite automaton, $\mathcal{M}_f = \pi(f)$, returned by Algorithm 1, can be constructed in polynomial time in #I, $|f|$ and $|Q_{f'}|$, where $\tau(f) = (\mathcal{A}, Q_{f'}, \Delta_{f'}, S_{f'}, F_{f'})$.* $\square$

The following theorem shows that the language induced by the output of a trail query f is equal to the regular language accepted by the intersection of the finite automaton representing f and the finite automaton representing the Hypertext database H, which constitutes the input to f.

Theorem 5.10 $\mathcal{L}(f(H)) = \mathcal{L}(\mathcal{M}_H \cap \mathcal{M}_f)$*, where H is a Hypertext database and f is a trail query.*

Proof. $(\mathcal{L}(f(H)) \subseteq \mathcal{L}(\mathcal{M}_H \cap \mathcal{M}_f))$: By Lemma 5.3 $\mathcal{L}(f(H)) \subseteq \mathcal{L}(\mathcal{M}_H)$. Thus, we need to show that it is also the case that $\mathcal{L}(f(H)) \subseteq \mathcal{L}(\mathcal{M}_f)$.

Let $\rho(T) \in \mathcal{L}(f(H))$ implying by Definition 4.11 that T $\models f$. We note that the constraint corresponding to (V1), i.e. that no two unique variables, which are distinct, can be mapped by an interpretation to the same T[i], and the constraint corresponding to (V2), i.e. that an interpretation maps the same variable to the same i $\in \{1,\ldots,\#I\}$ are both satisfied by Definition 4.3 of an interpretation σ.

We show that $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_f)$ by induction on the length of f. Recall that we have assumed that all conditions in f have the same length of one.

(*Basis*): If $|f| = 1$, then f is a condition, say C. The result follows by Definition 4.7 part (T1) and the description of τ in Definition 5.7 part (1), since there is a transition in $\mathcal{M}_f$ for each page number i_j with $\sigma(\mathrm{Z}) = i_j$.

(*Induction*): Assume that the result holds when $|f| = \mathrm{k}$, where $\mathrm{k} \geq 1$; we then need to prove that the result holds when $|f| > \mathrm{k}$.

We consider the four cases according to the structure of f as follows:

1. If f is the query $g_1 \wedge g_2$, then by Definition 4.7 part (T2) $\mathrm{T} \models f$ iff $\mathrm{T} \models g_1$ and $\mathrm{T} \models g_2$. Furthermore, by the inductive hypothesis, $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{g_1})$ and $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{g_2})$. Therefore by the description of τ in Definition 5.7 part (2), $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the intersection of the intermediate automata $\mathcal{M}_{g_1'}$ and $\mathcal{M}_{g_2'}$. The result follows from Algorithm 1 which implies that $\mathcal{M}_f$ is equal to the intersection of $\mathcal{M}_{g_1}$ and $\mathcal{M}_{g_2}$.

2. If f is the query $\Diamond(g)$, then by Definition 4.7 part (T3) $\mathrm{T} \models f$ iff $\exists$ i $\in \{0,\dots,\#\mathrm{T}-1\}$ such that $T^i \models g$. Furthermore, by the inductive hypothesis, $\rho(T^i) \in \mathcal{L}(\mathcal{M}_g)$. Therefore by the description of τ in Definition 5.7 part (3), $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the concatenation of $\mathcal{M}_\gamma$ and the intermediate automaton $\mathcal{M}_{g'}$. The result follows from Algorithm 1 which implies that $\mathcal{M}_f$ is equal to the concatenation of $\mathcal{M}_\gamma$ and $\mathcal{M}_g$.

3. If f is the query $\bigcirc(g)$, then by Definition 4.7 part (T4) $\mathrm{T} \models f$ iff $\#\mathrm{T} > 1$ and $T^1 \models g$. Furthermore, by the inductive hypothesis, $\rho(T^1) \in \mathcal{L}(\mathcal{M}_g)$. Therefore by the description of τ in Definition 5.7 part (4), $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the concatenation of $\mathcal{M}_\bigcirc$ and the intermediate automaton $\mathcal{M}_{g'}$. The result follows from Algorithm 1 which implies that $\mathcal{M}_f$ is equal to the concatenation of $\mathcal{M}_\bigcirc$ and $\mathcal{M}_g$.

4. If f is the query $\triangle(g)$, then by Definition 4.7 part (T5) $\mathrm{T} \models f$ iff $\#\mathrm{T} > 0$ and $T^i \models g$, where i $= \#\mathrm{T}-1$. Furthermore, by the inductive hypothesis, $\rho(T^i) \in \mathcal{L}(\mathcal{M}_g)$. Therefore by the description of τ in Definition 5.7 part (5), $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the concatenation of $\mathcal{M}_\gamma$ and the intermediate automaton $\mathcal{M}_{g'}$. Moreover, when contructing $\mathcal{M}_{g'}$ $\triangle$flag will have been set to *true* implying that $\rho(T^i) \in \mathcal{L}(\mathcal{M}_{g'})$, where $i = \#\mathrm{T}-1$. The result follows from Algorithm 1 which implies that $\mathcal{M}_f$ is equal to the concatenation of $\mathcal{M}_\gamma$ and $\mathcal{M}_g$.

$(\mathcal{L}(\mathcal{M}_H \cap \mathcal{M}_f) \subseteq \mathcal{L}(f(H)))$: Let $w \in \mathcal{L}(\mathcal{M}_H \cap \mathcal{M}_f)$, where $w = \delta(s_1)\delta(s_2)\dots\delta(s_n)$ (see Definition 5.2). Thus $w = \rho(\mathrm{T})$ for some trail T in R, with $\#\mathrm{T} = \mathrm{n}$, since $w \in \mathcal{L}(\mathcal{M}_H)$. Furthermore, there exists an interpretation σ over H such that $\forall$ i $\in \{1,\dots,\mathrm{n}\}$, $\forall$ Z $\in \varphi(\mathrm{cond}(s_i, \delta(s_i), s_{i+1}))$, $\sigma(\mathrm{Z}) = \delta(s_i)$, since by Algorithm 1 both conditions, (V1) and (V2), are satisfied by the path whose label is w.

We now show that $\mathrm{T} \models f$ with respect to H and σ, implying that $w \in \mathcal{L}(f(H))$ as required, by induction on the length of f. As before, recall that we have assumed that all conditions have the same length of one.

(*Basis*): If $|f| = 1$, then f is a condition, say C. The result follows as in the previous basis step.

(*Induction*): Assume that the result holds when $|f| = \mathrm{k}$, where $\mathrm{k} \geq 1$; we then need to prove that the result holds when $|f| > \mathrm{k}$.

We consider the four cases according to the structure of f as follows:

1. If f is the query $g_1 \wedge g_2$, then by the description of τ in Definition 5.7 part (2) $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{g_1'})$ and $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{g_2'})$. Moreover, by Algorithm 1, $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{g_1})$ and $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{g_2})$. Furthermore, by the inductive hypothesis, $\mathrm{T} \models g_1$ and $\mathrm{T} \models g_2$. The result then follows by Definition 4.7 part (T2).

2. If f is the query $\Diamond(g)$, then by the description of τ in Definition 5.7 part (3) $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the concatenation of $\mathcal{M}_\gamma$ and $\mathcal{M}_{g'}$. Moreover, by Algorithm 1, $\exists$ i $\in \{0,\dots,\#\mathrm{T}-1\}$ such that $\rho(T^i) \in \mathcal{L}(\mathcal{M}_g)$. Furthermore, by the inductive hypothesis, $T^i \models f$. The result then follows by Definition 4.7 part (T3).
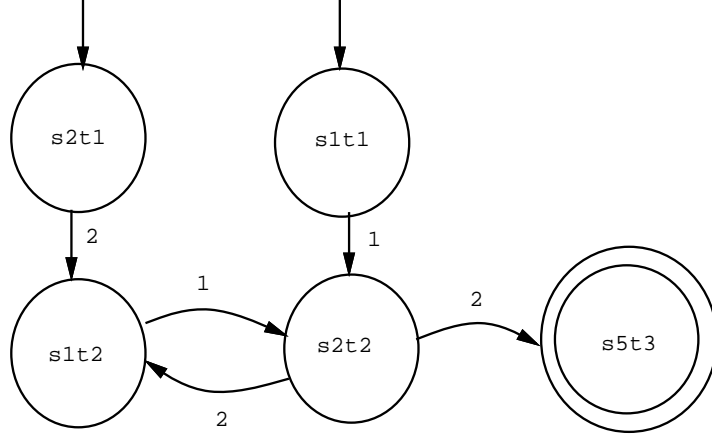
Figure 4: A finite automaton for the output of a trail query

3. If f is the query $\bigcirc(g)$, then by the description of τ in Definition 5.7 part (4) $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the concatenation of $\mathcal{M}_{\bigcirc}$ and $\mathcal{M}_{g'}$. Moreover, by Algorithm 1, $\#\mathrm{T} > 1$ and $\rho(T^1) \in \mathcal{L}(\mathcal{M}_g)$. Furthermore, by the inductive hypothesis, $T^1 \models f$. The result then follows by Definition 4.7 part (T4).

4. If f is the query $\triangle(g)$, then by the description of τ in Definition 5.7 part (5) $\rho(\mathrm{T}) \in \mathcal{L}(\mathcal{M}_{f'})$, where $\mathcal{M}_{f'}$ is the concatenation of $\mathcal{M}_{\gamma}$ and $\mathcal{M}_{g'}$. Moreover, when contructing $\mathcal{M}_{g'}$ $\triangle$flag will have been set to *true* implying that $\rho(T^i) \in \mathcal{L}(\mathcal{M}_{g'})$, where $i = \#\mathrm{T}-1$. By Algorithm 1 $\#\mathrm{T} > 0$ and $\rho(T^i) \in \mathcal{L}(\mathcal{M}_g)$, where i $= \#\mathrm{T}-1$. Furthermore, by the inductive hypothesis, $T^i \models f$. The result then follows by Definition 4.7 part (T5). □

Example 6 The finite automaton $\mathcal{M}_H \cap \mathcal{M}_f$, where $\mathcal{M}_H$ is shown in Figure 2 and $\mathcal{M}_f$ is shown in Figure 3, which accepts the same language that is induced by *f(H)*, is shown in Figure 4.

In order to test whether *f(H)* is nonempty, by Theorem 5.10 we can check whether $\mathcal{L}(\mathcal{M}_H \cap \mathcal{M}_f)$ is nonempty, which can be done in nondeterministic logspace [JONE75] and thus in polynomial time in the size of $\mathcal{M}_H \cap \mathcal{M}_f$.

The following corollary is an immediate consequence of Theorems 5.2, 5.8 and 5.10, since the class of regular star-free languages is closed under intersection.

Corollary 5.11 $\mathcal{L}(f(H))$ *is a star-free regular language.* □

We note that by Theorem 6.4 in [EMER90] PLTL is exactly as expressive as the class of star-free languages which do not include ϵ. Corollary 5.11, in view of Theorem 5.10, implies that $\mathcal{L}(f(H))$ is actually a member of a proper subclass of the class of star-free regular languages corresponding to the intersection of the star-free regular languages accepted by the finite automata representing trail queries and Hypertext databases, respectively.

The following corollary concerning the complexity of constructing $\mathcal{M}_H$, $\mathcal{M}_f$ and the intersection thereof is an immediate consequence of Definition 5.1, Lemma 5.6 and Theorem 5.9.

Corollary 5.12 *The following statements are true:*

1. $\mathcal{M}_H$ *can be constructed in linear time in $\#I$ and $|R|$.*

2. $\mathcal{M}_f$ *can be constructed in time exponential in the the number of conjunctions, between subformulae of f, plus one; if there are no conjunctions in f then $\mathcal{M}_f$ can be constructed in polynomial time in $\#I$ and $|f|$.*

22

3. $\mathcal{M}_H \cap \mathcal{M}_f$ can be constructed in polynomial time in $|Q_H|$ and $|Q_f|$, where Q_H and Q_f are the state sets of $\mathcal{M}_H$ and $\mathcal{M}_f$, respectively. □

It is interesting to compare Corollary 5.12 to a result of [LICH84] showing that although model checking of PLTL formulae (see Definition 6.1 in Section 6) is exponential time in the length of the formulae it is linear time in the size of the temporal structure being checked.

A special case worth considering is when R is acyclic. In this case the number of trails in R is finite and thus the regular language defined by the output of a query is a finite regular language and therefore trivially star-free. As we will see in the next section, the fact that R is acyclic does not necessarily reduce the complexity of computing the star-free regular language induced by $f(H)$, namely $\mathcal{L}(f(H))$.

Finally, it would be useful to extend HQL to enable the specification of second order notions such as specifying that only the shortest trails (i.e. the trails with the least count) would be included in $f(H)$. Obviously for such an extension further analysis of $\mathcal{M}_H \cap \mathcal{M}_f$ would have to be carried out.

6 The Complexity of Navigation in Hypertext

In this section we investigate the complexity of deciding whether a Hypertext database is a model of a trail formula. Our results show that, in general, this problem cannot be solved efficiently (assuming P $\neq$ NP). On the positive side we exhibit two significant subclasses of $\Diamond$-free trail formulae for which this decision problem can be solved in polynomial time.

Our results imply that the navigation problem is not easily solved in the general case. In practice Hypertext systems could include algorithms which return randomized and/or fuzzy solutions.

The following lemma shows that if a Hypertext database is a model of a given trail formula f then we can find a trail, which satisfies f, and whose count is no greater than #I multiplied by $|Sub(f)|$.

Lemma 6.1 *Let a Hypertext database, $H = <I, R>$, be a model of the trail formula, f. Then $\exists$ T in R such that $T \models f$ and $\#T \leq \#I \cdot |Sub(f)|$.*

Proof. Let L in R be a trail such that $L \models f$ holds; such a trail exists in R, since H is a model of f. We prove the result by showing that there exists a subtrail (see Definition 3.6), T of L, such that $T \models f$ and $\#T \leq \#I \cdot |Sub(f)|$; the proof is by induction on $|Sub(f)|$.
   (*Basis*): If $|Sub(f)| = 1$, then the result follows from Definition 4.7 part (T1), since #L > 0 and the prefix L_1 satisfies $L_1 \models f$ and $\#L_1 = 1$.
   (*Induction*): Assume that the result holds when $|Sub(f)| = $ k, where k $\geq$ 1; we then need to prove that the result holds when $|Sub(f)| = $ k+1.
   We conclude the result by considering the form of f according to the four cases, T2 to T5, of Definition 4.7 as follows (recalling the definition of the concatenation of two arrays given in Definition 3.3):

1. If f is of the form $f_1 \wedge f_2$, then $L \models f_1$ and $L \models f_2$. Furthermore, by the inductive hypothesis $T \models f_1$ and $T \models f_2$, where T is a subtrail of L, and $\#T \leq \#I \cdot \max(|Sub(f_1)|, |Sub(f_2)|)$. The result follows, since $\max(|Sub(f_1)|, |Sub(f_2)|) \leq |Sub(f)|$.

2. If f is of the form $\Diamond(f')$, then $\exists$ i $\in \{0,\ldots,\#L-1\}$ such that $L^i \models f'$. Furthermore, by the inductive hypothesis there exists a subtrail T of L^i, with $\#T \leq \#I \cdot |Sub(f')|$, such that T $\models f'$. The result follows by Proposition 4.2 part (8) which implies that $T \models \Diamond(f')$, since $|Sub(f')| \leq |Sub(f)|$.

3. If f is of the form $\bigcirc(f')$, then $\#L > 1$ and $L^1 \models f'$. Furthermore, by the inductive hypothesis there exists a subtrail T of L^1 such that T $\models f'$, with $\#T \leq \#I \cdot |Sub(f')|$. Let L_j be the prefix of L such that L_jT is also a prefix of L. It follows that there exists a subtrail T' of L in R such that $\#T' = 1$ and $T'[1] = L[j]$, since $\#L_j \geq 1$. The result follows, since by Definition 4.7 part (T4) T'T $\models f$ and $\#(T'T) = \#T + 1$.

4. If f is of the form $\triangle(f')$, then $\#L > 0$ and $L^i \models f'$, where i = $\#L-1$. The result follows, since by Definition 4.7 part (T5) $L^i \models \triangle(f')$ and $\#L^i = 1$. $\square$

Definition 6.1 (Model checking) The *model checking* problem is the problem of deciding whether a Hypertext database is a model of a trail formula.

We proceed to investigate the complexity of the model checking problem.

Lemma 6.2 *The model checking problem is in NP.*

Proof. In order to show that the model checking problem is in NP we present a nondeterministic polynomial time algorithm that decides the problem (cf. [SIST85, Theorem 3.5]).

Let $H = <I, R>$ be a Hypertext database and f be a trail formula. We first guess a trail T in R and an interpretation σ over H. It can easily be checked in polynomial time in $|f|$ that the restriction of σ to the unique variables in the subformulae of f is a one-to-one mapping. Furthermore, by Lemma 6.1 we can assume without loss of generality that $\#T \leq \#I \cdot |Sub(f)|$.

We now present an algorithm, which verifies whether T $\models f$ or not. The algorithm maintains a set, called label(i), for each marker i of T, which is initialized to the empty set. It then considers all of the subformulae $g \in Sub(f)$ in increasing order of $|g|$, assuming as before that all conditions in g have the same length of one, as follows:

1. If g is the condition C, with variable Z, then add g to label(i) iff $\sigma \models$ C and T[i] = σ(Z).

2. If g is the formula $g_1 \wedge g_2$, then add g to label(i) iff $g_1, g_2 \in$ label(i).

3. If g is the formula $\diamond(g')$, then add g to label(i) iff $\exists$ j $\in \{$i,...,$\#T\}$ such that $g' \in$ label(j).

4. If g is the formula $\bigcirc(g')$, then add g to label(i) iff $g' \in$ label(i+1).

5. If g is the formula $\triangle(g')$, then add g to label(i) iff $g' \in$ label($\#T$).

At the end of the above algorithm, whose time complexity is a polynomial in $\#I$ and $|f|$, it can be verified that T $\models f$ iff $f \in$ label(1). $\square$

Theorem 6.3 *The model checking problem is NP-complete.*

Proof. By Lemma 6.2 model checking is in NP. It remains to show that the problem is NP-hard.

In order to show NP-hardness we reduce, in polynomial time, 3SAT [GARE79] to the model checking problem using a reduction similar to that used in [SIST85, Theorem 3.5].

Let J = $J_1 \wedge \ldots \wedge J_m$ be a conjunction of m clauses on a finite set of variables $\{x_1, \ldots x_n\}$ such that $\forall$ i $\in \{1,\ldots,$m$\}$, $J_i = L_{i1} \vee L_{i2} \vee L_{i3}$ is a clause of three literals, where each literal L_{ij} is either x_q or $\neg x_q$, with q $\in \{1,\ldots,$n$\}$.

We define an operator, denoted by ϕ, which returns the clauses that contain a literal x_i or $\neg x_i$, as follows:

- $\phi(x_i) = \{J_j \mid x_i$ is a literal in $J_j\}$ and

- $\phi(\neg x_i) = \{J_j \mid \neg x_i$ is a literal in $J_j\}$.

Let $H(\mathrm{J}) = <\mathrm{I(J), R(J)}>$ be the Hypertext database defined by

1. The set of strings $\{a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_m, y_0, y_1, \ldots, y_n\} \subseteq \Sigma^*$, where the a_i's, b_i's, c_i's and y_i's are distinct, and $\mathrm{A} \in \mathcal{U}$.

2. $\#\mathrm{I(J)} = 3\mathrm{n}+1$.

3. $\forall\, \mathrm{i} \in \{1,\ldots,\mathrm{n}\}$, $\mathrm{I(J)[i]} = (\mathrm{A}, w_i)$, with $w_i = a_i c_1 \ldots c_k$, $\mathrm{k} \leq \mathrm{m}$, and where $\phi(x_i) = \{J_1, \ldots, J_k\}$.

4. $\forall\, \mathrm{i} \in \{\mathrm{n}+1,\ldots,2\mathrm{n}\}$, $\mathrm{I(J)[i]} = (\mathrm{A}, w_i)$, with $w_i = b_{i-n} c_1 \ldots c_k$, $\mathrm{k} \leq \mathrm{m}$, and where $\phi(\neg x_{i-n}) = \{J_1, \ldots, J_k\}$.

5. $\forall\, \mathrm{i} \in \{2\mathrm{n}+1,\ldots,3\mathrm{n}+1\}$, $\mathrm{I(J)[i]} = (\mathrm{A}, y_{i-(2n+1)})$.

6. $\mathrm{R(J)} = \{(\psi(y_{i-1}), \psi(w_i)), (\psi(w_i), \psi(y_i)), (\psi(y_{i-1}), \psi(w_{i+n})), (\psi(w_{i+n}), \psi(y_i)) \mid \mathrm{i} \in \{1,\ldots,\mathrm{n}\}\}$, where $\forall\, \mathrm{j} \in \{1,\ldots,\#\mathrm{I(J)}\}$, $\psi(\mathrm{I(J)[j]})$ denotes the page number j of $\mathrm{I(J)}$, $\psi(w_i)$ stands for $\psi((\mathrm{A}, w_i))$ and $\psi(y_i)$ stands for $\psi((\mathrm{A}, y_i))$.

The structure of $\mathrm{R(J)}$ is shown in Figure 5, where for notational convenience the page numbers $\psi((\mathrm{A},\, a_i c_1 \ldots c_k))$, $\psi((\mathrm{A},\, b_{i-n} c_1 \ldots c_k))$ and $\psi((\mathrm{A},\, y_{i-(2n+1)}))$ are abbreviated to a_i, b_i and y_i, respectively.
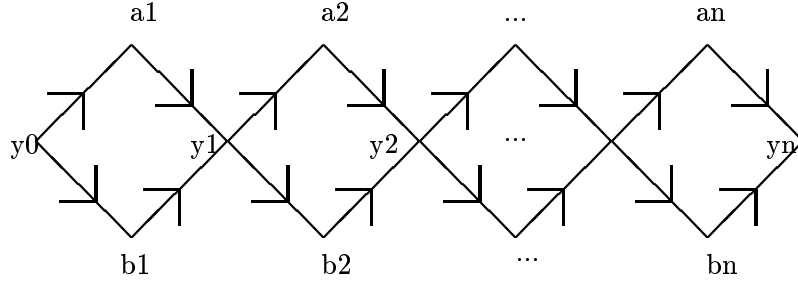

Figure 5: The reachability relation R(J)

It can now be verified that J is satisfiable iff $H(\mathrm{J})$ is a model of the trail formula,

$$substr(X_0, y_0) \wedge \Diamond(substr(X_1, c_1)) \wedge \ldots \wedge \Diamond(substr(X_m, c_m)) \wedge \triangle(substr(X_{m+1}, y_n)),$$

where $X_0, X_1, \ldots, X_m, X_{m+1}$ are m+2 distinct normal variables. $\quad\square$

The following result is an immediate consequence of the reduction in the proof of Theorem 6.3.

Corollary 6.4 *The model checking problem is NP-complete for the class of Hypertext databases having acyclic reachability relations and the class of $\bigcirc$-free trail formulae whose conditions do not contain any unique variables.* $\quad\square$.

The next result shows that the model checking problem does not become tractable by allowing unique variables in conditions and disallowing normal variables.

Theorem 6.5 *The model checking problem is NP-complete for the class of Hypertext databases having acyclic reachability relations and the class of $\bigcirc$-free trail formulae whose conditions do not contain any normal variables.*

Proof. As in Theorem 6.3, by Lemma 6.2 the model checking problem is in NP. It remains to show that the problem is NP-hard. In order to conclude the result we modify the reduction of 3SAT given in the proof of Theorem 6.3. Firstly, we modify parts 3 and 4 of the construction of $H(\mathrm{J})$ as follows:

3. $\forall$ i $\in$ {1,...,n}, I(J)[i] = (A, w_i), with $w_i = a_i$; these pages represent the literals x_i.

4. $\forall$ i $\in$ {n+1,...,2n}, I(J)[i] = (A, w_i), with $w_i = b_{i-n}$; these pages represent the literals $\neg x_i$.

Secondly, for each x_q, q $\in$ {1,...,n}, such that $\phi(x_q) = \{J_1, \ldots, J_k\}$ we add k additional pages to I(J). Each such additional page I(J)[i] is of the form (A, $a_i c_j$), j $\in$ {1,...,k}, where c_j corresponds to the clause J_j. Similarly, for each $\neg x_q$, q $\in$ {1,...,n}, such that $\phi(\neg x_q) = \{J_1, \ldots, J_k\}$ we add another k additional pages to I(J). As in the preceding case each such additional page I(J)[i] is of the form (A, $b_{i-n} c_j$), j $\in$ {1,...,k}, where c_j corresponds to the clause J_j.

As in the previous theorem the page numbers: $\psi((A, a_i))$, $\psi((A, b_{i-n}))$ and $\psi((A, y_{i-(2n+1)}))$ are abbreviated for notational convenience to a_i, b_i and y_i, respectively, with i $\in$ {1,...,n}. In addition, both the page numbers: $\psi((A, a_i c_j))$ and $\psi((A, b_{i-n} c_j))$ are abbreviated to c_j, with j $\in$ {1,...,k}, where the distinction between $a_i c_j$ and $b_{i-n} c_j$ is understood from context.

Moreover, for each page number i of I(J) representing x_i, we add a subgraph to the reachability relation R(J) of Theorem 6.3 (see Figure 5) as follows. Firstly, we add an arc from the page number i of I(J) to each of the page numbers c_j. Secondly, we add an arc from each of the page numbers c_j, j $\in$ {1,...,k}, to each of the page numbers c_p, with j < p and p $\in$ {1,...,k}. Finally, we add an arc from each of the page numbers c_j to the page number y_i. We repeat this process for each page number i of I(J) representing $\neg x_i$.

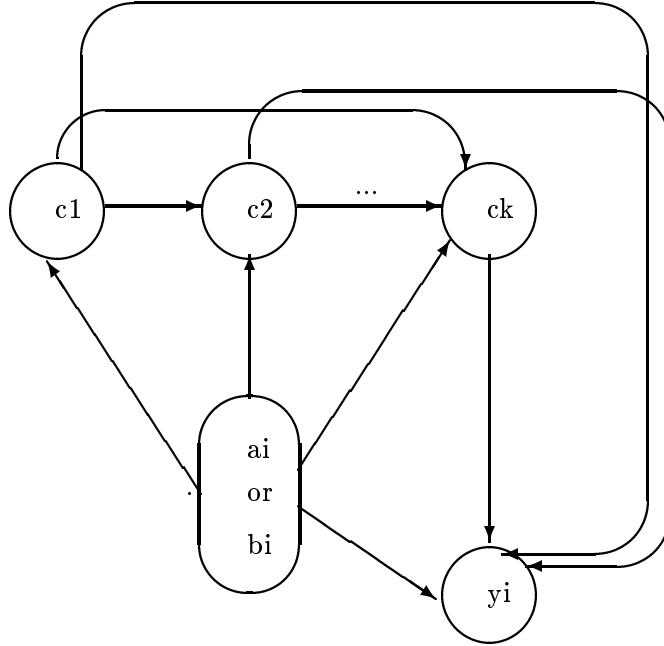The subgraph added to the reachability relation R(J) of Theorem 6.3 is shown in Figure 6.

Figure 6: The subgraph added to R(J)

It can now be verified that J is satisfiable iff H(J) is a model of the trail formula,

$$substr(Y_0, y_0) \wedge \Diamond(substr(Y_1, c_1)) \wedge \ldots \wedge \Diamond(substr(Y_m, c_m)) \wedge \triangle(substr(Y_{m+1}, y_n)), \quad (2)$$

where $Y_0, Y_1, \ldots, Y_m, Y_{m+1}$ are m+2 distinct unique variables.

Specifically, if H(J) is a model of (2) then for some trail T $\in$ R(J), T satisfies (2) with respect to some interpretation, σ over H. Now, for each $\sigma(Y_i)$, i $\in$ {1,...,m}, let a_i or b_i be the page number such that either $(a_i, \sigma(Y_i))$ or $(b_i, \sigma(Y_i))$ is in R(J). It follows therefore that J is satisfiable under an assignment where each literal corresponding to a_i or b_i is made true.

Correspondingly, if J is satisfiable then we need to exhibit a trail T that satisfies (2) with respect to some interpretation, σ over H. Now, since J is satisfiable there is an assignment that makes each J_j in J, $j \in \{1,\ldots,m\}$, true. Furthermore, J_j is true due to a literal, say L_i, being true in J_j. Let either a_i or b_i be the page number corresponding to L_i and let c_j be the page number corresponding to J_j such that either (a_i, c_j) or (b_i, c_j) is in R(J). Let σ be such that $\sigma(Y_0) = y_0$, $\sigma_{m+1}(Y_{m+1}) = y_n$ and for $j \in \{1,\ldots,m\}$ $\sigma(Y_j) = c_j$. The result now follows, since there is a trail passing through all the c_j's satisfying the desired trail formula. $\square$.

We next show that the model checking problem is also NP-complete for the subclass of $\diamond$-free trail formulae.

Theorem 6.6 *The model checking problem is NP-complete for the class of $\diamond$-free trail formulae.*

Proof. By Lemma 6.2 the model checking problem is in NP. It remains to show that the problem is NP-hard. In order to show NP-hardness we reduce, in polynomial time, the longest path problem, which is known to be NP-complete [GARE79], to the model checking problem for the class of $\diamond$-free trail formulae.

Let G = (V, E) be a directed graph with V = $\{v_1, \ldots, v_m\}$ and s, t $\in$ V. We need to solve the problem: does there exist a *simple path* (that is, a path in which no node occurs more than once) from s to t in G of length k or more?

We first let ϕ be a one-to-one mapping from V to $\{A\} \times \Sigma^*$, where A $\in \mathcal{U}$.

Next let $H(G) = <I(G) , R(G)>$ be a Hypertext database defined by

1. $\#I(G) = |V| = m$.

2. $\forall$ i $\in \{1,\ldots,m\}$, $\phi(v_i) = I(G)[i]$.

3. $\psi(G) = R(G)$, where ψ is an isomorphism from G to R(G) such that $\psi((v_i, v_j)) = (i, j)$, where i is the page number satisfying $\phi(v_i) = I(G)[i]$ and j is the page number satisfying $\phi(v_j) = I(G)[j]$.

It can now be verified that there exists a simple path from s to t in G of length k or more iff $H(G)$ is a model of the trail formula

$$substr(Y_0, \beta(\phi(s))) \wedge \bigcirc(att(Y_1, A)) \wedge \bigcirc^2(att(Y_2, A)) \wedge \\ \bigcirc^{k-1}(att(Y_{k-1}, A)) \wedge \triangle(substr(Y_k, \beta(\phi(t)))), \tag{3}$$

where $Y_0, Y_1, \ldots, Y_k$ are distinct unique variables and $\bigcirc^p$ denotes the composition, $\bigcirc \ldots \bigcirc$, p times, with p $\in \omega$. $\square$

The following two results follow from the reduction in the proof of Theorem 6.6.

Corollary 6.7 *The model checking problem is NP-complete for the class of $\diamond$-free trail formulae whose conditions do not contain any normal variables.* $\square$.

The next corollary is easily shown by replacing all occurrences of $\bigcirc$ in (3) by $\diamond$; it also follows from Theorem 6.5.

Corollary 6.8 *The model checking problem is NP-complete for the class of $\bigcirc$-free trail formulae whose conditions do not contain any normal variables.* $\square$.

We now show that model checking can be done in polynomial time for $\diamond$-free trail formulae when the reachability relation of the Hypertext database is acyclic. In order to prove the result we first solve the following graph-theoretic problem:

Given a directed acyclic graph (V, E) and a family $\{V_i\}$ ($i \in \{0,\ldots,k,\}$) of subsets of V, does there exist a path, $< v_0, \ldots, v_k >$, of length k in G such that $v_i \in V_i$?

Let us call this the *path of length k* problem. We note that when we consider directed acyclic graphs every path is actually a simple path.

We next show that a solution to the path of length k problem can be obtained in $O(\text{k} \cdot |E|)$ time; when (V, E) is acyclic then k $\leq$ |V|.

Lemma 6.9 *Given a directed graph (V, E) and a family $\{V_i\}$ ($i \in \{0,\ldots,k,\}$) of subsets of V, then the path of length k problem can be solved in $O(k \cdot |E|)$ time.*

Proof. In order to solve the problem we give the pseudo-code of an algorithm designated, PATH((V, E), $\{V_i\}$), which takes as input a directed graph and a family of k+1 subsets of V and returns YES if there is a solution to the path of length k problem, otherwise it returns NO.

For each subset $V_i \subseteq$ V we maintain an auxiliary set, called P_i, which stores all the nodes that can be reached from a node in V_0 via a path of length i. We also assume that adjacent(v) denotes the set of nodes: $\{$u | (v, u) $\in$ E$\}$.

Algorithm 2 (PATH((V, E), $\{V_i\}$))
1. **begin**
2. $P_0 := V_0$;
3.   **for** i = 1 **to** k **do**
4. $P_i := \emptyset$;
5.   **for all** v $\in P_{i-1}$ **do**
6.   **for all** u $\in$ adjacent(v) **do**
7.   **if** u $\in V_i$ **then**
8. $P_i := P_i \cup \{$u$\}$;
9.   **end if**
10.   **end for**
11.   **end for**
12.   **end for**
13.   **if** $P_k \neq \emptyset$ **then**
14.   **return** YES;
15.   **else**
16.   **return** NO;
17.   **end if**
18. **end.**

It can be verified that the algorithm is correct and solves the path of length k problem in $O(\text{k} \cdot |E|)$ time. $\square$

We next present a special case when model checking can be done in polynomial time.

Theorem 6.10 *The model checking problem can be solved in polynomial time in the length of the repository and the length of the trail formula, for the class of Hypertext databases having acyclic reachability relations and for the class of $\diamond$-free trail formulae.*

Proof. Let $H = <$I, R$>$ be a Hypertext database with R being acyclic and let f be a $\diamond$-free trail formula. Since f is $\diamond$-free it follows that we can use Proposition 4.2 in order to obtain a trail formula which is equivalent to f and in the following *normal form*:

$$C_{1_1} \wedge \ldots \wedge C_{n_1} \wedge \triangle(C_{1_2}) \wedge \ldots \wedge \triangle(C_{n_2}) \wedge \bigcirc \ldots \bigcirc (C_{1_3}) \wedge \ldots \wedge$$
$$\bigcirc \ldots \bigcirc (C_{n_3}) \wedge \bigcirc \ldots \bigcirc \triangle(C_{1_4}) \wedge \ldots \wedge \bigcirc \ldots \bigcirc \triangle(C_{n_4}),$$

where each C_i is a condition which is also a subformula of f, noting that for any trail T in R it cannot be the case that $T \models \triangle \bigcirc (f)$.

On using Proposition 4.2 it can be verified that if a trail formula, f, is $\diamond$-free, then f can be converted into a normal form $\diamond$-free trail formula in time polynomial in $|f|$. For the rest of the proof we assume that f is in normal form.

Let $\bigcirc^k[f]$ denote the set of conditions in the conjuncts of f having no instance of $\triangle$ and exactly k instances of $\bigcirc$, where $k \geq 0$, and let $\max\bigcirc(f)$ denote the maximum number of instances of $\bigcirc$ in any conjunct of f. In addition, let $\triangle[f]$ denote the set of conditions, $\{C_{1_2}, \ldots, C_{n_2}, C_{1_4}, \ldots, C_{n_4}\}$, in conjuncts of f having an instance of $\triangle$. Finally, let F be an array of sets of page numbers such that $\#F = \max\bigcirc(f) + 2$ and such that initially $\forall\, i \in \{1,\ldots,\#F\}$, $F[i] = \emptyset$.

The first step in our model checking algorithm is syntactic. If one of the following constraints is violated, then H is not a model of f, since a contradiction arises:

1. There do not exist two unique variables, which are distinct, contained in the set of conditions $\bigcirc^k[f]$ for any $k \geq 0$, i.e. no two unique variables, which are distinct, can appear at the *same time*.

2. There do not exist two unique variables, which are distinct, contained in the set of conditions $\triangle[f]$, i.e. no two unique variables, which are distinct, can appear at the *final time*.

3. There do not exist natural numbers, k_1, k_2, with $k_1 \neq k_2$ and $0 \leq k_1, k_2 \leq \max\bigcirc(f)$, such that the intersection of the set of variables contained in $\bigcirc^{k_1}[f]$ with the set of variables contained in $\bigcirc^{k_2}[f]$ is nonempty, i.e. no variable can appear in two distinct *times*.

4. $\forall\, k$, $0 \leq k < \max\bigcirc(f)$, the intersection of the set of variables contained in $\bigcirc^k[f]$ with the set of variables contained in $\triangle[f]$ is empty, i.e. no variable can appear before and at the *final time*.

It can be verified that all of the above constraints can be checked in polynomial time in $|f|$. Assuming that the above constraints are satisfied then the following statements are true:

1. $\forall\, k$, $0 \leq k \leq \max\bigcirc(f)$, if $\bigcirc^k[f]$ is nonempty, then we can assume that there exists only one variable in its conjuncts such that this variable does not appear in any other $\bigcirc^q[f]$, where $0 \leq k \neq q \leq \max\bigcirc(f)$, i.e. there is only one variable per *time unit*.

2. If $\triangle[f]$ is nonempty, then we can assume that there is only one variable in its conjuncts, such that this variable does not appear in any other set $\bigcirc^k[f]$, where $0 \leq k < \max\bigcirc(f)$, i.e. there is only one variable for the *final time unit*.

3. If the intersection of the set of variables contained in $\triangle[f]$ and the set of variables contained in $\bigcirc^k[f]$ is nonempty, where $k = \max\bigcirc(f)$, then the single variable assumed to be in their conjuncts must be the same variable, i.e. in this case the *final time* coincides with the time implied by $\max\bigcirc(f)$, otherwise we assume that their single respective variables are distinct.

We note that statements (1) and (2) above are valid, since if H is a model of f and T is a trail in R such that $T \models f$ with respect to some interpretation σ over H, then for the variables X_i in the conjunctions of $\bigcirc^k[f]$ or $\triangle[f]$, σ must map X_i to the same page number.

From now on we assume that the above statements are enforced in normal form trail formulae by a straightforward renaming of variables. We are not concerned whether the single variable per time unit is normal or unique due to the fact that R is acyclic and thus all the paths in the directed acyclic graph corresponding to R are simple.

For each page number, pn of I, we maintain an auxiliary set, called cond(pn), which is initialized to the empty set. We then consider all the conditions C, of all the conjuncts of f, in turn and add C to cond(pn) according to the following constraint:

- Add C to cond(pn) iff $\sigma \models$ C, assuming that σ is an interpretation over H and $\sigma(Z) =$ pn, where Z is the variable of C.

Constructing the sets cond(pn) can be done in polynomial time in $\#I$ and $|f|$.

Next, for each page number pn of I add pn to F according to the following two rules:

1. If $(\bigcirc^{k}[f] \neq \emptyset) \subseteq$ cond(pn), where $0 \leq k \leq \max\bigcirc(f)$, then add pn to F[k+1].

2. If $(\triangle[f] \neq \emptyset) \subseteq$ cond(pn), then add pn to F[$\#$F].

In the next step of our model checking algorithm we check if one of the ensuing constraints is violated in which case H is not a model of f.

1. For any k, $0 \leq k \leq \max\bigcirc(f)$, if $\bigcirc^{k}[f] \neq \emptyset$, then F[k+1] $\neq \emptyset$,

2. If $\triangle[f] \neq \emptyset$, then F[$\#$F] $\neq \emptyset$.

The above constraints can be tested in polynomial time in $\#I$ and $|f|$. We therefore assume that the above two constraints are satisfied.

In order to utilize Lemma 6.9, we modify F by setting F[i] to $\{1,\ldots,\#I\}$, $\forall$ i $\in \{1,\ldots,\#F\}$ whenever F[i] $= \emptyset$. The result now follows, on viewing R as a directed acyclic graph, and viewing the prefix F_{k+1} of F, where k+1 $= \#F-1$, as a family of k+1 subsets of I. The model checking problem thus reduces to the path of length k problem, with the following final test:

$P_{k}^{*} \cap$ F[$\#$F] $\neq \emptyset$, where P_{k}^{*} denotes the set of nodes, $P_{k} \cup \{$u $\mid$ v $\in P_{k}$ and (v, u) is an arc in the transitive closure of R$\}$.   □

The following corollary, which is an immediate consequence of Theorem 6.10, demonstrates a special case when model checking can be solved in polynomial time even when the reachability relation is cyclic.

Corollary 6.11 *The model checking problem can be solved in polynomial time in the length of the repository and the length of the trail formula, for the class of $\diamond$-free trail formulae whose conditions do not contain any unique variables and such that the normal variables appearing in conditions at distinct times are distinct.*

Proof. The result follows, since in this special case constraints (1) to (4) in the proof of Theorem 6.10 do not have to be satisfied. Furthermore, in this special case, Algorithm 2 will return YES iff there exists a trail in R, which is not necessarily loop-free, i.e. a node may appear more than once in the path corresponding to the trail.   □

The following corollary, which is given for completeness, follows immediately from Theorem 6.10 on using Algorithm 2, since we need only inspect $O(\#I^{2})$ pages.

Corollary 6.12 *The model checking problem can be solved in polynomial time in the length of the repository and the length of the trail formula, for the classes of $\diamond$-free and $\bigcirc$-free trail formulae.*
□

The next theorem shows that if we relax the condition stated in Corollary 6.11, namely that normal variables appearing in conditions at distinct times be distinct, then the model checking problem is again NP-complete.

Theorem 6.13 *The model checking problem is NP-complete for the class of $\diamond$-free trail formulae whose conditions do not contain any unique variables.*

Proof. By Lemma 6.2 the model checking problem is in NP. It remains to show that the problem is NP-hard. In order to show NP-hardness we reduce, in polynomial time, the clique [BUCK90] of size k problem, which is known to be NP-complete [GARE79], to the model checking problem for the class of $\diamond$-free trail formulae, whose conditions do not contain any unique variables.

Let G = (V, E) be a graph with V = $\{v_1, \ldots, v_m\}$. We need to solve the problem: does G contain a clique of size k or more, where k $\leq$ |V|?

We first let ϕ be a one-to-one mapping from V to $\{A\} \times \Sigma^*$, where A $\in \mathcal{U}$. Next, let $H(G) = $ <I(G), R(G)> be a Hypertext database defined by

1. #I(G) = |V| = m.

2. $\forall$ i $\in \{1, \ldots, m\}$, $\phi(v_i) = $ I(G)[i].

3. ψ(G) = R(G), where ψ is an isomorphism from G to R(G) such that $\psi(\{v_i, v_j\}) = \{$(i,j), (j,i)$\}$, where i and j are the page numbers satisfying $\phi(v_i) = $ I(G)[i] and $\phi(v_j) = $ I(G)[j], respectively.

Let K = $\{1, \ldots, k\}$ and lex(K) = <(1,2), (1,3), $\ldots$, (1,k), (2,3), $\ldots$, (2,k), $\ldots$, (k$-$1,k)> be the lexicographically ordered sequence [HALM74] of $(k^2 - k)/2$ pairs in K^2 such that (i, j) is in lex(K) iff i < j. Furthermore, let path(K) = <1,2,1,3, $\ldots$, 1,k,2,3, $\ldots$, 2,k, $\ldots$, k$-$1,k,1> be the sequence of the $(k^2 - k)+1$ numbers in K resulting from transforming each pair (i, j) in lex(K) into the subsequence i,j and then adding 1 at the end of the resulting sequence.

It can be verified that G contains a clique of size k or more iff we can find k nodes in the directed graph corresponding to R(G) numbered from 1 to k such that path(K) is a path in this directed graph.

We next transform path(K) into the following $\diamond$-free trail formula, denoted by f(path(K)), whose conditions do not contain any unique variables, namely

$$att(X_1, A) \wedge \bigcirc(att(X_2, A)) \wedge \bigcirc^2(att(X_1, A)) \wedge \bigcirc^3(att(X_3, A)) \wedge \ldots \wedge$$
$$\bigcirc^{2(k-1)-2}(att(X_1, A)) \wedge \bigcirc^{2(k-1)-1}(att(X_k, A)) \wedge \bigcirc^{2(k-1)}(att(X_2, A)) \wedge$$
$$\bigcirc^{2(k-1)+1}(att(X_3, A)) \wedge \ldots \wedge \bigcirc^{4(k-2)}(att(X_2, A)) \wedge \bigcirc^{4(k-2)+1}(att(X_k, A)) \wedge \ldots \wedge$$
$$\bigcirc^{(k^2-k)-1}(att(X_{k-1}, A)) \wedge \bigcirc^{k^2-k}(att(X_k, A)) \wedge \bigcirc^{(k^2-k)+1}(att(X_1, A)),$$

where $X_1, \ldots, X_k$ are distinct normal variables and $\bigcirc^k$ denotes the composition, $\bigcirc \ldots \bigcirc$, k times, with k $\in \omega$.

It can be verified that G contains a clique of size k or more iff $H(G)$ is a model of the trail formula f(path(K)). $\quad \square$

Table 1, shown below, summarizes the complexity results obtained in this section; the reader can verify that all possible cases have been considered. Yes in the acyclic column indicates that the reachability relation of the Hypertext database is acyclic, Yes in the no-normal column indicates that trail formulae do not contain normal variables, Yes in the no-unique column indicates that trail formulae do not contain unique variables, Yes in the $\diamond$-free column indicates that trail formulae are $\diamond$-free, Yes in the $\bigcirc$-free column indicates that trail formulae are $\bigcirc$-free, NPC stands for NP-complete, P stands for polynomial time; finally, No† in the no-normal column indicates that normal variables appearing in conditions at distinct times are distinct.

7   Concluding Remarks

An attempt has been made to tackle the navigation problem for Hypertext, namely the problem of getting "lost in hyperspace". In order to solve this problem we utilized temporal logic and defined the navigation semantics of Hypertext in terms of the query language HQL, which is based on a subset of PLTL. In the navigation semantics of HQL the notion of a trail is central. The output to

acyclic	no-normal	no-unique	$\diamond$-free	$\bigcirc$-free	complexity
Yes	No	Yes	No	Yes	NPC
Yes	Yes	No	No	Yes	NPC
No	Yes	No	Yes	No	NPC
No	No	Yes	Yes	No	NPC
Yes	No	No	Yes	No	P
No	No†	Yes	Yes	No	P
No	No	No	Yes	Yes	P

Table 1: Summary of the complexity results

a trail query with respect to a Hypertext database is the set of all trails in the Hypertext database which satisfy the trail query.

We investigated the evaluation of HQL queries, showing that by using the automata-theoretic approach we can construct a finite automaton representing the output $f(H)$ of a trail query f with respect to a Hypertext database H. We have shown in Corollary 5.11 that the language $\mathcal{L}(f(H))$ induced by $f(H)$ is a star-free regular language. Furthermore, in Corollary 5.12 we obtained an upper bound on the time complexity of the construction of $\mathcal{M}_H \cap \mathcal{M}_f$, which is exponential in the number of conjunctions, between subformulae of f, plus one.

We also investigated the complexity of the model checking problem for HQL queries and summarized our results in Table 1. We have shown that, in general, this problem is NP-complete but that there are important special cases when the problem can be solved in polynomial time.

Our conclusion is that the navigation problem in Hypertext cannot be solved efficiently unless users have some a priori knowledge about the order in which the pages of information are structured in the database graph. Having this knowledge users can utilise the $\diamond$-free polynomial-time subset of HQL to pursue a one-step at a time navigation session through the database graph. Experimental research has to be carried out in order to ascertain whether expensive queries will often be posed. In practice, it may also be useful to seek randomized and/or fuzzy solutions to this problem.

References

[BEER94] C. Beeri and Y. Kornatzky, A logical query language for hypermedia systems. *Information Sciences*, **77**, (1994), 1-37.

[BUCK90] F. Buckley and F. Harary, *Distance in Graphs*. Redwood City, CA., Addison-Wesley, 1990.

[BUSH45] V. Bush, As we may think. *Atlantic Monthly*, **76**, (1945), 101-108.

[CLAR86] 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*, **8**, (1986), 244-263.

[CONK87] J. Conklin, Hypertext: An introduction and survey. *IEEE Computer*, **20**, (1987), 17-41.

[CONS89] M.P. Consens and A.O. Mendelzon, Expressing structural Hypertext queries in GraphLog. In: *Proceedings of ACM Conference on Hypertext, Hypertext'89*, pp. 269-292, 1989.

[CODD79] E.F. Codd, Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, **4**, (1979), 379-434.

[EMER90] E.A. Emerson, Temporal and modal logic. In: *Handbook of Theoretical Computer Science, Volume B*, Ed. J. Van Leeuwen, Amsterdam, Elsevier Science Publishers B.V., 1990, Chapter 16, pp. 997-1072.

[ESPA94] J. Esparza and M. Nielson, Decidability issues for Petri nets. *Bulletin of the EATCS*, **52**, (1994), 245-262.

[FRIS92] M.F. Frisse and S.B. Cousins, Models for Hypertext. *Journal of the American Society for Information Science*, **43**, (1992), 183-191.

[GABB80] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, On the temporal analysis of fairness. In: *Proceedings of the ACM Symposium on Principles of Programming Languages*, pp. 163-173, 1980.

[GARE79] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, Freeman, 1979.

[HALA88] F.G. Halasz, Reflections on Notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, **31**, (1988), 836-852.

[HALM74] P.R. Halmos, *Naive Set Theory.* New York, Springer-Verlag, 1974.

[HOPC79] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Reading, MA., Addison-Wesley, 1979.

[JONE75] N.D. Jones, Space-bounded reducibility among combinatorical problems. *Journal of Computer and System Sciences*, **11**, (1975), 68-85.

[LICH84] O. Lichtenstein and A. Pnueli, Checking that finite state concurrent programs satisfy their linear specification. In: *Proceedings of the ACM Symposium on Principles of Programming Languages*, pp. 97-107, 1994.

[MCNA71] R. McNaughton and S. Pappert, *Counter-Free Automata.* Cambridge, MA., MIT Press, 1971.

[MANN92] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Berlin, Springer-Verlag, 1992.

[MEND95] A.O. Mendelzon and P.T. Wood, Finding regular simple paths in graph databases. *SIAM Journal on Computing*, **24**, (1995), 1235-1258.

[NYCE89] J.M. Nyce and P. Kahn, Innovation, pragmaticism, and technological continuity: Vannevar Bush's memex. *Journal of the American Society for Information Science*, **40**, (1989), 214-220.

[PETE81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, NJ, Prentice-Hall, 1981.

[PERR90] D. Perrin, Finite automata. In: *Handbook of Theoretical Computer Science, Volume B*, Ed. J. Van Leeuwen, Amsterdam, Elsevier Science Publishers B.V., 1990, Chapter 1, pp. 1-57.

[RESC71] N. Rescher and A. Urquhart, *Temporal Logic.* Wien, Springer-Verlag, 1971.

[RIVL94] E. Rivlin, R. Botafogo and B. Shneiderman, Navigating in hyperspace: Designing a structure-based toolbox. *Communications of the ACM*, **37**, (1994), 87-96.

[SCHN88]   J.L. Schnase, J. Leggett, C. Kacmar and C. Boyle, A comparison of Hypertext systems. Research Report 88-017, Hypertext Research Lab., Texas A&M University, 1988.

[SEDG90]   R. Sedgewick, *Algorithms in C.* Reading, MA., Addison-Wesley, 1990.

[SIST85]   A.P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logics. *Journal of the ACM*, **32**, (1985), 733-749.

[STOT89]   P.D. Stotts and R. Furata, Petri-net-based Hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems*, **7**, (1989), 3-29.

[STOT92]   P.D. Stotts, R. Furata and J.C. Ruiz, Hyperdocuments as automata: Trace-based browsing property verification. In: *Proceedings of the ACM Conference on Hypertext, ECHT'92*, pp. 272-281, 1992.

[THOM90]   W. Thomas, Automata on infinite objects. In: *Handbook of Theoretical Computer Science, Volume B*, Ed. J. Van Leeuwen, Amsterdam, Elsevier Science Publishers B.V., 1990, Chapter 4, pp. 133-191.

[TOMP89]   F.W.M Tompa, A data model for flexible Hypertext database systems. *ACM Transactions on Information Systems*, **7**, (1989), 85-100.

[TREN73]   M. Trenchard Jr., Axioms and theorems for a theory of arrays. *IBM Journal of Research and Development*, **17**, (1973), 135-175.

[ULLM88]   J.D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. I.* Rockville, MD., Computer Science Press, 1988.

[VAND88]   A. Van Dam, Hypertext '87 keynote address. *Communications of the ACM*, **31**, (1988), 887-895.

[VARD86a]   M.Y. Vardi and P. Wolper, Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, **32**, (1986), 183-221.

[VARD86b]   M.Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification. In: *Proceedings of the IEEE Symposium on Logic in Computer Science*, pp. 332-344, 1986.