In IEE Proceedings on Software Engineering, 144(3), 175-184, June 1997.

A Formal Framework for Hypertext Systems*

Mark d'Inverno and Mark Priestley
School of Computer Science,
University of Westminster,
115 New Cavendish Street,
London, W1M 8JS, UK
{dinverm,priestm}@westminster.ac.uk

Michael Luck
Department of Computer Science,
University of Warwick,
Coventry
CV4 7AL, UK
mikeluck@dcs.warwick.ac.uk

Abstract

Hypertext and hypertext systems are seeing a remarkably rapid growth in both use and development. In this paper we aim to consolidate on previous work by presenting a specification that captures the essential abstractions of hypertext systems. We argue that this specification provides a framework for hypertext systems in that it provides explicit and unambiguous definitions of hypertext terms, an explicit environment for the presentation, comparison and evaluation of hypertext systems, and a foundation for future research and development in the field.

1 Introduction

Many formal reference models of hypertext have been presented in the literature [1, 14, 20, 25, 26] but, whilst these models give valuable theoretical insights into certain aspects of the structure of hypertext, they are not in themselves adequate vehicles for the presentation, evaluation and comparison of different systems. In this paper we describe an approach to the formal specification of hypertext systems which allows the development of a common conceptual framework and provides an environment in which to discuss, design, develop and evaluate them.

We have argued elsewhere [22] that a formal framework must satisfy three distinct requirements, described below.

1. It must precisely and unambiguously provide meanings for common concepts and terms and do so in a readable way. A common conceptual framework will then exist if there is a generally held understanding of the salient issues involved.

2. It should enable alternative designs of particular models and systems to be explicitly presented, compared and evaluated.

3. It should provide a foundation for subsequent development of new and increasingly more refined concepts. In particular, practitioners should be able to choose the level of abstraction suitable for their purpose.

* An earlier version of this paper has appeared as [11].

The use of formal concepts allows explicit and unambiguous descriptions of terms and complex systems to be given. In our model, the Z specification language [24], in particular, is used as a means of formalisation for a number of reasons. First, the language is based upon primitive mathematical concepts such as set theory and first order logic, making it accessible to researchers from a variety of different backgrounds. Second, it is expressive enough to allow a consistent, formal and unified account of a system and its associated operations. Third, we have considerable experience of the benefits of constructing formal models using Z in a number of fields, including interactive conferencing systems [6], distributed artificial intelligence [5], multi-agent systems [9, 10] and design methodologies [8]. In particular, we have found Z appropriate for building *formal frameworks*, necessary to enable a rigorous approach for any discipline [6, 8, 21, 22].

Such frameworks are provided through well-structured specifications. These *describe* systems at their highest level of abstraction, with complexity being added at each successive lower level, allowing irrelevant information to be removed from consideration. As a result, different modular components of a system can be isolated and described separately, and commonalities in different parts of a system can be recognised and presented as such. Abstraction renders prejudice about design unnecessary; consequently, a specification of a *general system* can be written. Indeed, Z schema boxes are ideal for manipulation in the design process since, viewing in many cases the design process as a constraint of possible states, design strategies can be presented as predicates in the appropriate state schemas. Such structured specifications, we argue, are tools which enable a more systematic analysis of hypertext.

An earlier attempt in the literature to provide a formal specification of a 'general' hypertext system, known as the Dexter Model of Hypertext [17], also used Z, with the motivation — to capture formally and informally the important abstractions found in a wide range of existing and future hypertext systems — similar in some respects to our own. The Dexter Model essentially comprises a collection of components — links and nodes — with an *accessor* function that maps a unique identifier to a node and a *resolver* function that maps descriptions of components to the components themselves. On top of this base, operations of adding, modifying and retrieving components are specified.

However, the Dexter Model specification is often obtuse and over-complicated, and only the most experienced Z practitioner with a good knowledge of hypertext would be able to gain much from it. Part of the problem is that there is no structuring of the specification. Instead, it starts with a large collection of *given sets* and introduces many concepts and functions before the first state schema is actually presented. In this sense, the specification is *flat* and does not aid the reader in building up a picture of the model of a hypertext. Moreover, the specification describes hypertext at a very low level of detail and is hence much more oriented towards implementation concerns. We argue that the immediate complexity does not serve the typical hypertext practitioner in providing an accessible model that can commonly be adopted by the hypertext community, and that the lack of abstraction mechanisms within the *flat* specification does not provide a framework in which to present and develop ideas concerning the design of hypertext systems.

A second effort at dealing with this problem is based around the Hypertext Abstract Machine (HAM) [3], which is a general-purpose server for a hypertext storage system. HAM has several features similar to our model, including nodes, links, graphs and attributes, and it describes the way information is represented before being used in information retrieval. The motivation of the work is to lay the foundations for a standard terminology in the development of hypertext technology. Though an important addition to the field, the model is only concerned with

problems of storage, and not with representing an information retrieval session. In addition, the model is not formal, and consequently does not provide the precision of a mathematical specification. Our work, by contrast, is both formal and, we argue, sufficiently expressive to provide a framework within which to detail *all* aspects of hypertext.

The specification of our framework is split into four parts. The first of these, in Section 2, presents what we believe is the most straightforward and intuitive description of a model of hypertext systems where nodes are treated as given sets and links as pairs of system nodes. The second part in Section 3 defines a mechanism for representing and designing history mechanisms. The third part, presented in Section 4, builds on the basic model and increases the level of specification detail in order to describe the internal details of a node. Both Section 2 and Section 4 are divided into three subsections: the first defines the structure of the system, the second defines the state of the system as it is being read, and the third presents a description of the basic applications of hypertext, namely how it facilitates structured movement through a particular information space. The last part, in Section 5, outlines how the model can be used to detail other features and applications of hypertext. Finally, Section 6 provides a summary of the paper and details current and future work.

2 The Basic Hypertext

2.1 Structure

At its highest level of abstraction, a hypertext system consists of a collection of basic elements. Typically, these elements are called *nodes*, but the name can vary from system to system. For example, in NoteCards they are *cards* [16], in KMS they are *frames* [2], in Intermedia they are called *documents* [27] and in Augment they are known as *statements* [13]. In the specification that follows, we choose to refer to these elements as *nodes*, as it is the most common term.

We start by defining the *given set* of nodes, and specifying that the contents of hypertext are just a collection of nodes.

$$[\mathsf{NODE}]$$

CONTENTS

$$Nodes : \mathbb{P}\,\mathsf{NODE}$$

On closer examination of hypertext systems, however, a rich and sophisticated structure is revealed. In particular, between nodes there exist certain connections, known as *links*, each suggesting some relationship between the nodes they connect. A *link* is a directional connection, pointing from one node (sometimes referred to as the *parent* node) to another node (sometimes referred to as the *child* node). It is therefore characterised by the nodes it connects, and can be defined as such. At this level of representation, links are not physical constructs, but are just connectivity relations between nodes (though we can add properties to links by elaborating this representation as shown, for example, in Section 5.1).

$$\mathsf{LINK} == \mathsf{NODE} \times \mathsf{NODE}$$

LINKS

$$Links : \mathbb{P}\,\mathsf{LINK}$$

Consequently, we define a hypertext system as a collection of nodes and links, where links must point *from* an existing system node. However, it is not necessary for a link to point *to* a system node; many hypertext systems include the notion that some links only have the potential to point to such a node. For example, the URL of an anchor in a WWW page could refer to a non-existent page. This is recognised in the hypertext system definition below.

HYPERTEXT

$$\begin{array}{l} CONTENTS \\ LINKS \\ \hline \mathrm{dom}\ Links \subseteq Nodes \end{array}$$

2.1.1 Button Nodes

Though the definitions above cover the basic structure of hypertext, there are often some special nodes that may be reached without using a link. We call such nodes *button* nodes. Furthermore, there may also be a default *starting* node for hypertext when a system is first used in a session. These special nodes are specified in the schema below, where a $StartNode$ is defined to be optional. (Definitions of the non-standard optional, and related concepts, may be found in Appendix A).

ButtonHYPERTEXT

$$\begin{array}{l} HYPERTEXT \\ Buttons : \mathbb{P}\,NODE \\ StartNode : \mathrm{optional}\ [NODE] \\ \hline Buttons \subseteq Nodes \\ StartNode \subseteq Nodes \\ StartNode \subseteq Buttons \end{array}$$

2.1.2 Typed Links

A second useful specialisation of the basic structure relates to links. In some hypertext systems, links may be grouped into *Link Functions*, as with the Up and $Next$ functions of the *emacs* INFO system. Rather than specifying a link the user instead specifies a function which isolates an appropriate link.

$$LINKFUNCTION == NODE \pfun NODE$$

TypedLINKS

$$\begin{array}{l} LINKS \\ LinkFunctions : \mathbb{P}\,LINKFUNCTION \\ \hline \bigcup LinkFunctions \subseteq Links \end{array}$$

As an extra constraint, a particular system might insist that every link should belong to a function but that a link must not belong to more than one function. In such a case, we would simply include the following predicate in the above schema.

$$\mathrm{setdisjoint}\ LinkFunctions \ \land \ \bigcup LinkFunctions = Links$$

2.2 The State of the Hypertext

Now that we have specified the *structure* of hypertext within this simple model, we must specify its *state* as it is used during an information retrieval session. All of the systems that we have investigated make use of the notion of the *position* of a user within the information space, and the *history* of that user's information retrieval session. A history provides a record of the nodes visited by a user in a session and, possibly, the way in which they were visited. Though there may be several distinct histories maintained in a system for different purposes, we initially consider the simplest kinds of browsing history, defined below by $StandardHistory$, the sequence of all the nodes visited (including repeats), and $Visited$, the set of nodes that have been visited. (A more detailed analysis of history mechanisms is provided in the next section.)

SimpleHISTORY

$$StandardHistory : \mathsf{seq}\ \mathsf{NODE}$$
$$Visited : \mathbb{P}\,\mathsf{NODE}$$
$$Visited = \mathsf{ran}\ StandardHistory$$

Now, we represent a user session by the hypertext, the user's history and current position within the information space.

HYPERTEXTState

HYPERTEXT
SimpleHISTORY
$$CurrentNode : \mathsf{NODE}$$
$$CurrentNode \in Nodes$$

As mentioned previously, the buttons that become available during a session might be dependent on the session itself. Here, the variable $RunButtons$, a superset of $Buttons$, represents those nodes that can currently be visited without the use of a link. In particular, it is typical that any previously visited node can be re-visited without using a link.

ButtonHYPERTEXTState

HYPERTEXTState
ButtonHYPERTEXT
$$RunButtons : \mathbb{P}\,\mathsf{NODE}$$
$$(\mathsf{defined}\ StartNode) \land (Visited \neq \{\}) \Rightarrow head\ StandardHistory = \mathsf{the}\ StartNode$$
$$Buttons \subseteq RunButtons$$
$$RunButtons \subseteq Nodes$$
$$Visited \subseteq RunButtons$$

2.3 Applications

One of the benefits of our use of the Z specification language is that the various operations provided in a system can be specified within the same formal framework. This property is not shared by many of the mathematical models presented in the literature; for example, Tompa [26] uses hypergraphs to give a formal account of the structure of a hypertext, but then specifies

the operations of reading the hypertext using a mixture of pseudo-code and informal English description. By contrast, a unified specification as provided here using Z, facilitates the exploration of, and reasoning about, the properties of operations and their effects on the state of the system in a formal manner.

Continuing with the specification, we state that an operation in an information retrieval session will not alter the actual linked structure of the hypertext.

ΔHYPERTEXTState

$$\text{HYPERTEXTState}$$
$$\text{HYPERTEXTState}'$$
$$\Xi\text{HYPERTEXT}$$

Starting a hypertext session changes the state of the hypertext by resetting the history.

Login

$$\Delta\text{HYPERTEXTState}$$

$$StandardHistory' = \langle\rangle$$

However, the operation depends on whether or not there is a defined start node for the system. If no start node is defined, then it must be supplied by the user.

LoginStartNode

$$\text{Login}$$
$$\Xi\text{ButtonHYPERTEXT}$$

$$\text{defined } StartNode$$
$$CurrentNode' = \text{the } StartNode$$

LoginNoStartNode

$$\text{Login}$$
$$\Xi\text{ButtonHYPERTEXT}$$
$$startnode? : \text{NODE}$$

$$\text{undefined } StartNode$$
$$startnode? \in Nodes$$
$$CurrentNode' = startnode?$$

At this point we can show how hypertext is used in an information retrieval session by moving through an information space using the hypertext system. Essentially, a hypertext system supports two types of moves: first, a user may move from one node to another by means of a link from their current node to a related node; second, if a user has some knowledge of a node, because it is a button node or a previously-visited node, for example, they may move directly to it without using a link.

The specification is structured by considering the properties that we require of a general move operation first, before giving details of a particular move. This provides an example of how Z has enabled us to modularise this specification and thus present the model in levels of increasing detail. In particular, it shows how the small schemas defined in our specification can be combined to define more sophisticated and complex states and operations. In general, a move

operation will alter the state of the hypertext: the current node may change and the history may change, but the actual linked structure of the hypertext will remain unchanged. In addition, a move operation may return a message to the user.

Move

ΔHYPERTEXTState

$message!$: optional [ERROR]

We can now distinguish between successful and unsuccessful attempts to move. A successful move will update the history list, appending the node that has just been visited, and there will be no message.

MoveOk

Move

$Visited' = Visited \cup \{CurrentNode\}$

$StandardHistory' = StandardHistory \frown \langle CurrentNode \rangle$

undefined $message!$

In contrast, failed moves leave the state of the hypertext unchanged, but require an error message to be given.

MoveFail

Move

ΞHYPERTEXTState

---

defined $message!$

Moving from the current node to another node using a link can then be described as specified in the following schema.

FollowLinkOk

$link$: LINK

MoveOk

$CurrentNode' = second\ link$

Such a move can be made in one of two ways given by the schemas UserChoosesLinkOk and UserChoosesLinkFunctionOk given below. In the first case user identifies the link, $link?$, itself and in the second case the user identifies a link function, $LinkFunction?$, intended to isolate an appropriate link.

UserChoosesLinkOk

FollowLinkOk$[link?/link]$

$link? \in Links \ \wedge \ first\ link? = CurrentNode \ \wedge \ second\ link? \in Nodes$

FollowLinkFunctionOk

FollowLinkOk
TypedLINKS
$LinkFunction?$: LINKFUNCTION

$LinkFunction? \in LinkFunctions$
$CurrentNode \in (\text{dom } LinkFunction?)$
$link = (CurrentNode, LinkFunction?\ CurrentNode)$

UserChoosesLinkFunctionOk $==$ FollowLinkFunctionOk $\setminus\ (link)$.

Moves not using links can be made using buttons by supplying as input the node to be visited.

MoveButtonOk

MoveOk
ΔButtonHYPERTEXTState
ΞButtonHYPERTEXT
$button?$: NODE

$button? \in RunButtons$
$CurrentNode' = button?$

Moving back to the previously visited node requires no input node. In the following schema the history is not updated.

MoveBack

MoveOk

$StandardHistory \neq \langle\rangle$
$CurrentNode' = last\ StandardHistory$

An alternative way to update the history can be specified as follows where the last node in the history is removed.

MoveBackAlternative

Move

$StandardHistory \neq \langle\rangle$
undefined $message!$
$CurrentNode' = last\ StandardHistory$
$StandardHistory = front\ StandardHistory$

This subsection has shown how, using the simple history mechanisms, structured movement is facilitated through the information space. Clearly, however, there are a number of other, different possibilities for updating the history. If we are to fulfill the requirements of a framework as a device to enable the presentation of particular hypertext systems, we require a much more general history mechanism within which these different mechanisms can be accommodated, considered next.

3 A History Mechanism Framework

In general, a hypertext system comprises a set of histories, each representing a different view of the current information retrieval session of a user. In order for each of these histories to maintain a coherent view, they must each be updated according to a well-defined strategy that depends on the type of move made by the user. For example, in the soft-link hypertext model proposed by Hu [19], the user's history is only updated if a link is used. If a button move is made then the history is not updated.

Now, in order to specify the kind of operations that will need to be performed on history lists we must first define some generic schemas that specify useful functions on sequences. Below, the function, $Prefixes$, takes a sequence and returns the set of all prefix sequences. $RestrictToLast$ takes an element and a set of sequences, and returns all those sequences that contain the element at the end of the sequence. Finally, the $Longests$ and $Shortests$ functions are each applied to a set of sequences and return all those sequences that have a length greater than or equal to, and less than or equal to, respectively, all others in the set.

$$
\begin{array}{|l}
\hline
[X] \\
Prefixes : \mathsf{seq}\ X \rightarrowtail \mathbb{P}(\mathsf{seq}\ X) \\
RestrictToLast : (X \times \mathbb{P}(\mathsf{seq}\ X)) \rightarrow \mathbb{P}(\mathsf{seq}\ X) \\
Longests : \mathbb{P}(\mathsf{seq}\ X) \rightarrow \mathbb{P}(\mathsf{seq}\ X) \\
Shortests : \mathbb{P}(\mathsf{seq}\ X) \rightarrow \mathbb{P}(\mathsf{seq}\ X) \\
\hline
\forall\, x : X;\ s : \mathsf{seq}\ X;\ ss : \mathbb{P}(\mathsf{seq}\ X) \bullet \\
\qquad Prefixes\ s = \{t : \mathsf{seq}\ X \mid t \subseteq s \bullet t\} \land \\
\qquad RestrictToLast(x, ss) = \{s : ss \mid x = last\ s \bullet s\} \land \\
\qquad Longests\ ss = \{s : ss \mid (\neg\ (\exists\, t : ss \bullet (\#t) > (\#s))) \bullet s\} \land \\
\qquad Shortests\ ss = \{s : ss \mid (\neg\ (\exists\, t : ss \bullet (\#t) < (\#s))) \bullet s\} \\
\hline
\end{array}
$$

We can now specify the operations that may be used to update a particular history in which the key component data type is a sequence of nodes referred to as a *history list*. Below, we specify six such operations. The $TruncLast$ and $TruncFirst$ functions respectively truncate the history list at the first occurrence of the *last-visited node* and the last occurrence of the *last-visited node*. Add adds the *last-visited node* to the history list, while $Leave$ leaves a history list unchanged. The function, $NoRepeat$, only adds a node to a history if the node is not already in the history and, finally, $NoRepeatShunt$ behaves like $NoRepeat$ except that when it appends a node to a history list, the first node in the history is removed. Although this list of operations is not comprehensive, and more are possible, it should be clear to the reader how other, particular functions for maintaining histories, may be specified.

In these definitions we use the following type abbreviation.

$$\mathsf{HISTORYOP} == (\mathsf{NODE} \times \mathsf{seq}\ \mathsf{NODE}) \rightarrow \mathsf{seq}\ \mathsf{NODE}$$

$TruncFirst, TruncLast, Add, Leave, NoRepeat, NoRepeatShunt$: HISTORYOP

$\forall\, x$: NODE; s : seq NODE $\bullet$

$TruncLast(x, s) =$

$(\mu\, t$: seq NODE $\mid \{t\} = Longests\,(RestrictToLast\,(x, Prefixes(s)))) \land$

$TruncFirst(x, s) =$

$(\mu\, t$: seq NODE $\mid \{t\} = Shortests\,(RestrictToLast\,(x, Prefixes(s)))) \land$

$Add(x, s) = s \frown \langle x \rangle \land$

$Leave(x, s) = s \land$

$x \in \mathsf{ran}\ s \Rightarrow NoRepeat\,(x, s) = s \land NoRepeatShunt\,(x, s) = s \land$

$x \notin \mathsf{ran}\ s \Rightarrow NoRepeat\,(x, s) = \langle x \rangle \frown s \land NoRepeatShunt\,(x, s) = \langle x \rangle \frown front\ s$

Every type of move that needs to be distinguished is given a *move tag* for just that purpose.

[MOVETAG]

In general, a history can be defined by its history list, a function that states which operation (as defined above) should be used to update the history and, optionally, a pointer to some node in the history list. Note that the function that identifies the operation to be used to update the history list is not only applied to the type of move but also to the length of the history (to take into account bounded lists).

History

$List$: seq NODE

$WhichOp$: $\mathbb{N} \rightarrow$ MOVETAG $\rightarrow$ HISTORYOP

$Pointer$: optional $[\mathbb{N}]$

$Length$: $\mathbb{N}$

---

defined $Pointer \Rightarrow$ the $Pointer \in$ dom $List$

$Length = \#List$

As an example of a particular history, consider the $Recent$ history list in HyperCard, which enables the user to display up to the last 42 cards visited (with no duplicates). We specify this particular mechanism in the next schema. This schema states that the length of the history is never greater than 42, that the history does not contain duplicates, and that there is no pointer In addition, if the length of the list is less than 42, then the $WhichOp$ function always selects the $NoRepeat$ operation, but if the length is exactly 42 then it selects the $NoRepeatShunt$ operation.

HyperCardRecent

History

$Length \leq 42$

$Length = \#(\mathsf{dom}\ List)$

undefined $Pointer$

$Length < 42 \Rightarrow (\forall\, m$: MOVETAG $\bullet\ WhichOp\ Length\ m = NoRepeat)$

$Length = 42 \Rightarrow (\forall\, m$: MOVETAG $\bullet\ WhichOp\ Length\ m = NoRepeatShunt)$

As already stated, a hypertext system may, in general, maintain a set of histories, each representing a different view of the users's information session, and updated according to a distinct strategy, specified by the local $WhichOp$ function. If $Visited$ is taken to be the set of nodes that

the user has visited (as previously) then we can specify the constraint that each history list must contain only nodes visited by the user.

GeneralHISTORY

$$Histories : \mathbb{F}\,\mathsf{History}$$
$$Visited : \mathbb{F}\,\mathsf{NODE}$$
$$\forall\, h : Histories \bullet \mathsf{ran}\ h.List \subseteq Visited$$

We now provide a general definition of how, given a node and a move tag, a history is updated, defined below as $Update$. Here, the history list is updated by applying the operation identified by the $WhichOp$ function to the node and the current history list. The other variables of a history are unchanged.

$$Update : \mathsf{NODE} \rightarrow \mathsf{MOVETAG} \rightarrow \mathsf{History} \rightarrow \mathsf{History}$$
$$\forall\, n : \mathsf{NODE};\ mt : \mathsf{MOVETAG};\ h, h' : \mathsf{History} \bullet$$
$$Update\ n\ mt\ h = h' \Leftrightarrow h'.List = (h.WhichOp\ h.Length\ mt)\,(n, (h.List)) \land$$
$$h'.WhichOp = h.WhichOp \land$$
$$h'.Pointer = h.Pointer$$

After every move, each individual history is updated in this way.

UpdateHistory

$$\Delta\mathsf{GeneralHISTORY}$$
$$mt : \mathsf{MOVETAG}$$
$$node : \mathsf{NODE}$$
$$Histories' = \{h, h' : \mathsf{History} \mid (h \in Histories) \land h' = Update\ node\ mt\ h \bullet h'\}$$
$$Visited' = Visited \cup \{node\}$$

For example, one of these histories might be updated using the Add operation every time a node is visited using an organisational link. This particular subgraph can then be presented to the user to show the complete path that a user has taken through the hypertext.

Note that we have not stated how pointers are updated in this specification. To do so, we can specify extra operations as, for example, given by the operation schema below, $MovePointer$, which simply moves the pointer to a node in the history list.

$MovePointer$

$$new? : \mathbb{N}$$
$$\Delta\mathsf{History}$$
$$new? \leq Length$$
$$\mathsf{the}\ Pointer' = new?$$
$$List' = List$$

We can then model pointer-driven data structures such as the Netscape $Back$ and $Forward$ operations in which moving back and forward through the history decrements or increments the pointer value by 1, respectively. In addition, whenever a new node is accessed using a link in Netscape, nodes occurring after the pointer in the history list are truncated.

$TruncatePointerList$

ΔHistory

$List' = (1 \mathinner{..} (\text{the } Pointer)) \lhd List$

$\text{the } Pointer' = \text{the } Pointer + 1$

In this section, we have not provided extensive specifications of particular history mechanisms, and instead focussed on the key framework components, illustrating them with examples. A more extensive treatment in a similar spirit is provided by Dix and Mancini [12], who compare several mechanisms in more detail.

4 The Highlight Hypertext

In order to model the fact that a node can contain certain hypertext elements, we now lower the level of description of a hypertext system. These hypertext elements are references, typically taking the form of highlighted text, which can then serve as the destination of, or source for, hypertext links. Many hypertext systems facilitate not only links connecting nodes, but regions within nodes.

4.1 Structure

Each node has a (possibly empty) set of internal hypertext references which we call *highlights*.

[HIGHLIGHT]

In general a node contains a set of highlights.

HighlightSetNODE

$NodeHighlights : \mathbb{P}$ HIGHLIGHT

One possible model for the above description is to represent this set as a sequence. If, in addition, there are no repeated highlights then an *injective* sequence can be used. As we shall see in Section 4.3, this model then supports the operations of scrolling backwards and forwards through these highlights.

HighlightNODE

$Highlights$: iseq HIGHLIGHT

The inside of each node then consists of highlights.

HighlightNODES

CONTENTS

$GetHighlights$: NODE $\pfun$ HighlightNODE

dom $GetHighlights \subseteq Nodes$

Next, we extend the notion of a link so that it can point to highlights within a node. We use the categories of Conklin [4] who differentiates two categories of link by drawing a distinction between *organisational* and *referential* links. We use these categories and introduce a third type which can be found in current hypertext systems, known as *span* links.

Organisational links Many hypertexts have an underlying structure, either as a consequence of the information space itself, or of the way in which information space is required to be presented to a user. *Organisational* links capture this underlying structure. For example, they may be hierarchical in nature so that there could be a standard way within the hypertext of moving from a given node to a *parent*, *child* or *sibling* node.

Referential links *Referential* links are typically non-hierarchical. They connect a *highlight*, which can be a point or a region within a node, to another node. Referential links are motivated by the *content* of a node, rather than by the underlying structure of the hypertext or information space.

Span links We define *span* links to be links which connect a *highlight* within a node to a *highlight* within another node. The notion of a cross-reference, for example, could be modelled in this fashion.

In this specification, we differentiate between these three types of link: we call organisational links *orglinks*, referential links *reflinks* and span links *spanlinks*. Some other mathematical models have had problems defining different kinds of link. Garzotto *et al.* [15] describe this ability to distinguish different kinds of link as an "innovative feature". In order to specify this more detailed hypertext, we must define a new type to represent links between highlights, and define each of the three link categories as subtypes.

HighlightLINK

$$From, To : \mathsf{NODE}$$
$$FromHighlight, ToHighlight : \mathsf{optional}\ [\mathsf{HIGHLIGHT}]$$

OrgLINK

HighlightLINK

$$\mathsf{undefined}\ FromHighlight \land \mathsf{undefined}\ ToHighlight$$

RefLINK

HighlightLINK

$$\mathsf{defined}\ FromHighlight \land \mathsf{undefined}\ ToHighlight$$

SpanLINK

HighlightLINK

$$\mathsf{defined}\ FromHighlight \land \mathsf{defined}\ ToHighlight$$
$$From \neq To$$

We may wish to reason about these links in terms of the nodes that they connect without concern for the kind of link. In order to do this, we introduce a function which maps our new representation of links to our old representation.

$$RecoverLink : \mathsf{HighlightLINK} \rightarrow\!\!\!\!\!\rightarrow \mathsf{LINK}$$
$$RecoverLinks : \mathbb{P}\,\mathsf{HighlightLINK} \rightarrow\!\!\!\!\!\rightarrow \mathbb{P}\,\mathsf{LINK}$$

$$\forall\, c : \mathsf{HighlightLINK};\ cs : \mathbb{P}\,\mathsf{HighlightLINK} \bullet$$
$$RecoverLink\ c = (c.From, c.To) \land$$
$$RecoverLinks\ cs = RecoverLink(\!|\ cs\ |\!)$$

The set of all links of the hypertext can now be given, and the two representations of organisational link within the model can be related.

HighlightLINKS

LINKS

$OrgLinks : \mathbb{P}\,\mathsf{OrgLINK}$

$RefLinks : \mathbb{P}\,\mathsf{RefLINK}$

$SpanLinks : \mathbb{P}\,\mathsf{SpanLINK}$

$HighlightLinks : \mathbb{P}\,\mathsf{HighlightLINK}$

$HighlightLinks = OrgLinks \cup RefLinks \cup SpanLinks$

$Links = RecoverLinks\ OrgLinks$

Our new model of hypertext is then provided by the following schema which ensures that all links are well defined.

HighlightHYPERTEXT

HYPERTEXT

HighlightLINKS

HighlightNODES

$$\forall\, l : HighlightLinks \bullet (l.From \in Nodes) \land (l.FromHighlight \subseteq \mathsf{ran}\ (GetHighlights\ l.From).Highlights)$$

Typing the links is similar to that given in the basic model, but the definition of what constitutes a link function is slightly different. A link function is any set of links for which no two links have the same *from-node* and *from-highlight*. In other words, there is only one way to leave a given position given a particular link function. Furthermore, we assert that a typed link function will only contain links which are either all organisational, all referential, or all span.

$$\mathsf{HlightLINKFUNCTION} == \{xs : \mathbb{P}\,\mathsf{HighlightLINK} \mid (\forall\, x, y : \mathsf{HighlightLINK} \bullet ((x \in xs) \land (y \in xs) \land (x \neq y)) \Rightarrow (x.From, x.FromHighlight) \neq (y.From, y.FromHighlight)) \bullet xs\}$$

TypedHighlightLINKS

HighlightLINKS

$OrgLinkFuns, RefLinkFuns, SpanLinkFuns : \mathbb{P}\,\mathsf{HlightLINKFUNCTION}$

$\bigcup OrgLinkFuns \subseteq OrgLinks$

$\bigcup RefLinkFuns \subseteq RefLinks$

$\bigcup SpanLinkFuns \subseteq SpanLinks$

As before, if we additionally require that no link of a particular subtype (say *span*) belonged to two functions, and that every link of a certain type (say *span* again) belonged to a function, we would write the following predicate.

$$\mathsf{setdisjoint}\ SpanLinkFunctions \land \bigcup SpanLinkFunctions = SpanLinks$$

4.2 State

Given the structure of the hypertext, we can define the position of a user within it. This not only includes the current node and history from the state of the basic model, but also the position of a user within a node. Such a position will be either defined, in which case the user will be positioned at some highlight, or undefined, which occurs, for example, when a node has no highlights or an organisational link has just been used to move to the current node.

HighlightHYPERTEXTState

HYPERTEXTState
HighlightHYPERTEXT
$Position$: optional [HIGHLIGHT]
HighlightNODE

θHighlightNODE $= GetHighlights\ CurrentNode$
$Position \subseteq (\text{ran}\ Highlights)$

4.3 Applications

A change in the state will not affect the structure.

ΔHighlightHYPERTEXTState

HighlightHYPERTEXTState
HighlightHYPERTEXTState$'$
ΞHighlightHYPERTEXT

Joining the hypertext is easily defined in terms of the basic model.

HighlightLogin

Login
ΔHighlightHYPERTEXTState

Any move using highlights may affect the state.

HighlightMove

ΔHighlightHYPERTEXTState
Move

We distinguish between two types of move. *Internal* moves involve the user scrolling through or selecting one of the highlights of the current node. *External* moves, by contrast, involve the user taking a link or moving to a button node. An internal move will not affect the state of the basic hypertext — the current node and history are not altered — and can only be made if the current node actually contains highlights.

InternalMoveOk

HighlightMove
ΞHYPERTEXTState
MoveOk

$Highlights \neq \langle\rangle$

There are three basic internal moves within a node. The first two, moving to the *next* highlight and moving to the *previous* highlight, both require the current position to be defined. The third move simply involves moving to a *chosen* highlight. For a definition of CycleNext and CyclePrevious, see Appendix A.

NextHighlight

InternalMoveOk

---

defined $Position$

the $Position' =$ CycleNext $((\text{the } Position), Highlights)$

PreviousHighlight

InternalMoveOk

---

defined $Position$

the $Position' =$ CyclePrevious $((\text{the } Position), Highlights)$

SelectHighlight

InternalMoveOk

$highlight?$: HIGHLIGHT

$highlight? \in (\text{ran } Highlights)$

the $Position' = highlight?$

The general *external* move may affect the position and the current node. Note, however, that the use of buttons is not changed in any way in this more sophisticated model.

ExternalMoveOk

HighlightMove

MoveOk

ΞHighlightHYPERTEXT

To use a link successfully, the parent node of the link must necessarily be the current node. If we use an organisational link then this is sufficient; they can be used from any position within a node. However, if we use a span link or referential link then the link must have a *from-highlight* equal to the current position. We re-use our basic model definition as follows.

FollowHighlightLinkOk

ExternalMoveOk

FollowLinkOk

$highlightlink$: OrgLINK

$highlightlink.From = CurrentNode$

$highlightlink \in OrgLinks \vee (highlightlink \in (RefLinks \cup SpanLinks) \wedge$
$Position = highlightlink.FromHighlight)$

$link = (highlightlink.From, highlightlink.To)$

$Position' = highlightlink.ToHighlight$

Just as with the top level representation of a link, the user chooses either the highlight link or a highlight link function intended to isolate the required highlight link. This would be specified in exactly the same way as in the UserChoosesLinkOk and UserChoosesLinkFunctionOk schemas of Section 2.3 for the top level link representation, so the extra schemas are omitted here.

5 Extensions

Although many formal models of hypertext have been proposed in the literature, there is still little consensus about what a definitive model should be. Indeed, one might argue that, in view of the rapid progress of the technology, it is probably premature to attempt a definitive formalisation. However, a significant benefit of using Z that we have discovered in this respect is that it does not restrict the specifier to any particular mathematical model; rather it provides a general mathematical framework within which different models, and even particular systems, can be defined and contrasted.

This claim is justified here by considering a number of more sophisticated features of hypertext and showing how the model defined in the previous sections can be elaborated to define and describe features and extensions found in a variety of hypertext systems.

5.1 Types and Values

A node or link may have a collection of types with associated values that may be used to structure the state space. In this case, the hypertext can be structured so that only certain types of links have access to certain types of nodes. In the following schema, we assert that for any two nodes connected by a link, the link and the nodes must have an associated type in common.

$$[\mathsf{TYPE}, \mathsf{VALUE}]$$
$$\mathsf{TYPEVALUEPAIRS} == \mathbb{P}(\mathsf{TYPE} \times \mathsf{VALUE})$$

TypedHYPERTEXT

$$\mathsf{HighlightHYPERTEXT}$$
$$NodeType : \mathsf{NODE} \rightarrow \mathsf{TYPEVALUEPAIRS}$$
$$HighlightLinkTypes : \mathsf{HighlightLINK} \rightarrow \mathsf{TYPEVALUEPAIRS}$$

$$\mathsf{dom}\ NodeType = Nodes$$
$$\mathsf{dom}\ HighlightLinkTypes = HighlightLinks$$
$$\forall c : HighlightLinks \mid c.To \in Nodes \bullet$$
$$(\exists t : \mathsf{TYPE} \bullet (t \in \bigcap \{first(\!| HighlightLinkTypes\ c |\!),$$
$$first(\!| NodeType\ (c.To) |\!), first(\!| NodeType\ (c.From) |\!)\}))$$

This mechanism can enable more effective information retrieval. For example, links of a particular type could be traversed in an order based on the values related to that type. Suppose that an information space contained information concerning various aspects of a city. Through appropriate use of typing, several distinct traversals of the hypertext might be possible, each relating to different aspects such as travel information, museums and galleries, or economic data, for example.

5.2 Specifying Different Topologies

The *topology* of a hypertext describes the way in which the nodes are connected. In the simplest case, a hypertext is seen as being a mere directed graph, but other organisations are possible to facilitate the successful movement though an information space. Van Dyke Parunak [23] provides a survey of possible topologies and, as an example, we define a hypertext which has a

$hierarchical$ structure. This is defined in terms of the organisational link function, $Parent$, and a set of organisational link functions called $Children$.

HierarchicalHYPERTEXT

HighlightHYPERTEXT
TypedHighlightLINKS
ButtonHYPERTEXT
$Parent$: HlightLINKFUNCTION
$Children$: $\mathbb{P}$ HlightLINKFUNCTION

$Parent \in OrgLinkFuns$
$Children \subseteq OrgLinkFuns$
defined $StartNode$
ran $(RecoverLinks\,(\bigcup Children)) = Nodes \setminus StartNode$
$(RecoverLinks\ Parent)^{-1} = RecoverLinks\,(\bigcup Children)$

5.3 User Navigation

A number of proposals have been made for defining *paths* through a document [25] as a means for helping a user navigate through it. A path would offer a reader a pre-defined route through a subset of the document, thus enabling an overview of the hypertext, or of a particular subject to be presented. A path can be just a sequence of nodes. Equally, it can take the form of a sequence of highlight links which actually take the user through particular highlights in the hypertext.

SIMPLEPATH $==$ seq NODE
PATH $==$ seq HighlightLINK

SimplePATHS

CONTENTS
$SimplePaths$: $\mathbb{P}$ SIMPLEPATH

---

ran $(\bigcup SimplePaths) \subseteq Nodes$

PATHS

HighlightLINKS
$paths$: $\mathbb{P}$ PATH

$\forall\, p : paths \bullet (\forall\, l, m : \text{HighlightLINK} \mid \langle l, m\rangle \text{ in } p \bullet$
$(l.To, l.ToHighlight) = (m.From, m.ToHighlight))$

5.4 Content of Nodes

So far in this specification, we have not considered the text that is stored at each node. In elaboration of this basic model, we define TextHYPERTEXT which includes a mapping from nodes to identifiers, and a mapping from identifiers to text. Thus, if text is altered then all nodes that map to the associated identifier of the text will map to the altered text. In this way, we allow

for the possibility of different nodes sharing the same content. Indeed, such an organisation has been cited as an advantage of the hypergraph model by Tompa [26]. The predicate part of the schema simply states that all nodes point to some text.

$[CHAR]$
$[TEXT\_ID]$
$STRING == \mathsf{seq}\ CHAR$
$TEXT == [text : STRING]$

TextHYPERTEXT

$HYPERTEXT$
$Text : NODE \pfun TEXT\_ID$
$Text\_Val : TEXT\_ID \pfun TEXT$

$\mathsf{dom}\ Text = Nodes$
$\mathsf{ran}\ Text \subseteq \mathsf{dom}\ Text\_Val$

Once a notion of content has been defined, it is then possible to define a further class of move operations that Conklin calls *keyword links* [4]. A simple example of this would be to search for all nodes containing a given string, and a successful operation for it is described in the schema below, where the input, $keyword?$, is the search string, and the set of nodes in the document containing the string is returned in the set, $found!$. One of these found nodes may then become the current node.

KeywordSearch

$TextHYPERTEXT$
$keyword? : STRING$
$found! : \mathbb{P}\, NODE$

$found! = \{n : Nodes \mid keyword? \in (Text\_Val(Text\ n)).text \bullet n\}$

5.5 Properties of Hypertext

It is straightforward in this framework to define additional properties of hypertext, and we describe two examples here. The first additional property, that of *accessibility*, states that every node in the hypertext can be reached from the start node of a user session (whether user or system defined) using organisational links only. The second property enforces the constraint that no nodes are *dead ends* or, equivalently, that there is always at least one organisational, referential or span link out of any node.

Accessibility

$ButtonHYPERTEXT$

$\mathsf{defined}\ StartNode \Rightarrow Nodes \subseteq \mathsf{ran}\ (StartNode \lhd Links\star)$
$\mathsf{undefined}\ StartNode \Rightarrow (\forall\, n : Nodes \bullet Nodes \subseteq \mathsf{ran}\ (\{n\} \lhd Links\star))$

NoDeadEnds

$HighlightHYPERTEXT$
$TypedHighlightLINKS$

$Nodes \subseteq \mathsf{dom}\ (RecoverLinks\ HighlightLinks)$

5.6 Authoring

In addition to operations for reading an information space using hypertext, many systems also provide *authoring* operations by means of which nodes and links can be added to a hypertext to represent the information space in a more effective way. For example, a link to another node is added in the WWW by the including a URL in an existing node. Adding a new node is known as indexing, and adding a new link is known as hyperization. Some example operations for authoring, which enable us to specify the addition of an organisational link (AddOrgLink) are specified below.

AddLink

ΔHYPERTEXT
ΞCONTENTS
$link?$: LINK

$link? \notin Links$
$first\ link? \in Nodes$
$Links' = Links \cup \{link?\}$

Adding a link to a highlight hypertext depends on the type of link that is being added. The common aspects of these operations are given in the following schema.

AddHighlightLink

ΔHighlightHYPERTEXT
$highlightlink?$: HighlightLINK

$highlightlink? \notin HighlightLinks$
$highlightlink?.From \in Nodes$
$highlightlink?.FromHighlight \subseteq$
$\quad$ (ran $((GetHighlights\ highlightlink?.To).Highlights)$)

Then, the operation of adding an organisational link, for example, is specified as follows. The operations of adding referential and span links are defined similarly.

AddOrgLink

AddHighlightLink
AddLink

$highlightlink? \in$ OrgLINK
$OrgLinks' = OrgLinks \cup \{highlightlink?\}$
$RefLinks' = RefLinks$
$SpanLinks' = SpanLinks$
$link? = (highlightlink?.To, highlightlink?.From)$

6 Summary

In this paper we have presented a specification that defines a formal framework for hypertext systems. This specification addresses all three concerns identified at the beginning of the paper as being of fundamental importance in the construction of such formal frameworks, in the

context of the hypertext community. The first requirement of providing precise definitions of relevant terms and concepts in a readable way has been achieved by means of the well-known Z specification language. In addition, the resulting specification provides a strong and clear base enabling alternative designs of particular models and systems to be explicitly presented, compared and evaluated. Finally, the specification provides a foundation for subsequent development of new and increasingly more refined concepts and, as stated earlier, it should, in particular, allow practitioners to choose the level of abstraction suitable for their purpose. For example, the specification has provided the foundation upon which to build a formal model of a new intelligent hypertext system [7, 18, 19]. This system uses statistical information collected in information retrieval sessions over a period of time to learn how best to aid the user in navigating through a given information space. Since our framework specification is well structured, we were able to choose the appropriate level of abstraction relevant to our purpose of modelling this new system. In this case, the learning model is concerned only with organisational links between nodes, and so we have developed the formalisms of the learning techniques within our most abstract model of hypertext.

The framework has been applied to produce specifications of existing hypertext systems including HyperCard and the World Wide Web and, as a result, we have been able to systematically evaluate and compare these systems. Further work continues in developing a more general version of the specification as we apply it to an increasing number of existing hypertext systems.

Acknowledgements

Thanks to Claire Cohen, Jennifer Goodwin, Paul Howells, Mike Hinchey, Michael Hu and Colin Myers for comments on earlier versions of this paper. The anonymous referees were especially helpful in providing comments and suggestions that have significantly improved the paper, and in identifying some errors. The specification contained in this paper was checked for correctness using the *f*UZZ package.

References

[1] F. Afrati and C. Koutras. A hypertext model supporting query mechanisms. In *Hypertext: Concepts, Systems and Applications. Proceedings of the European Conference on Hypertext*, pages 52–66, 1990.

[2] R. Akscyn, D. McCracken, and E. Yoder. KMS: A distributed hypertext for managing knowledge in organizations. *Communications of the ACM*, 31(7), July 1988.

[3] B. Campell and J. Goodman. HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM*, 31(7):856–861, 1988.

[4] J. Conklin. Hypertext: An Introduction and Survey. *Computer*, 20(9):17–41, 1987.

[5] M. d'Inverno. Using Z to capture the essence of a contract net. Master's thesis, Programming Research Group, Oxford University, 1988.

[6] M. d'Inverno and J. Crowcroft. Design, specification and implementation of an interactive conferencing system. In *Proceedings of IEEE Infocom, Miami, USA. Published IEEE*, 1991.

[7] M. d'Inverno and M. J. Hu. A Z specification of the soft-link hypertext model. In M. G. Hinchey, editor, *ZUM'97: 10th International Conference of Z Users, Lecture Notes in Computer Science*, pages 297–316, 1997.

[8] M. d'Inverno, G. R. Justo, and P. Howells. A formal framework for specifying design methodologies. *Software Process: Improvement and Practice*, 2(3):181–195, September, 1996.

[9] M. d'Inverno and M. Luck. A formal view of social dependence networks. In *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian Workshop on Distributed Artificial Intelligence, Lecture Notes in Artificial Intelligence, 1087*, pages 115–129. Springer Verlag, 1996.

[10] M. d'Inverno and M. Luck. Formalising the contract net as a goal directed system. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 72–85. Springer-Verlag, 1996.

[11] M. d'Inverno and M. Priestley. Structuring a Z specification to provide a unifying framework for hypertext systems. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: 9th International Conference of Z Users, Lecture Notes in Computer Science*, pages 83–102, Heidelberg, 1995. Springer-Verlag.

[12] A. Dix and R. Mancini. Specifying history and backtracking mechanisms. In P. Palanque and F. Paterno, editors, *Formal Methods in Human-Computer Interaction*. Springer-Verlag, To appear, 1997.

[13] D. Englebart. Authorship provisions in Augment. In *Proceedings of the IEEE COMPCON*, Spring 1984.

[14] P. Garg. Abstraction Mechanisms in Hypertext. *Communications of the ACM*, 31(7):862–870, 1988.

[15] F. Garzotto, P. Paolini, and D. Schwabe. HDM–A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, 11(1):27–50, 1993.

[16] F. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM*, 31(7), July 1988.

[17] F. Halasz and M. Schwartz. The Dexter Hypertext. *Communications of the ACM*, 37(2):30–39, 1994.

[18] M. J. Hu. *An Intelligent Information System.* PhD thesis, UCL, 1994.

[19] M. J. Hu and P. Kirstein. An Intelligent Hypertext System. In *The First International Workshop on Intelligent Hypertext (CIKM-93)*, Washington, November 1993.

[20] D. Lange. A formal model for hypertext. In *Proceedings of the NIST Hypertext Standardization Workshop*, 1990.

[21] M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press / MIT Press, 1995.

[22] M. Luck and M. d'Inverno. Structuring a Z specification to provide a formal framework for autonomous agent systems. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: 9th International Conference of Z Users, Lecture Notes in Computer Science*, pages 48–62. Springer-Verlag, 1995.

[23] H. Van Dyke Parunak. Hypermedia Topologies and User Navigation. In *Hypertext '89 Proceedings*, 1989.

[24] J. M. Spivey. *The Z Notation*. Prentice Hall, Hemel Hempstead, 2nd edition, 1992.

[25] D. Stotts and R. Furata. Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. *ACM Transactions on Information Systems*, 7(1):3–29, 1989.

[26] F. Tompa. A Data Model For Flexible Hypertext Database Systems. *ACM Transactions on Information Systems*, 7(1):85–100, 1989.

[27] N. Yankelovich, B. Haan, N. Meyrowitz, and S. Drucker. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer*, 1988.

A Z Extensions

We have found it useful in a specification to be able to assert that an element is optional. For example, in the specification given in this paper, the error message returned by a hypertext move is optional. If the move is unsuccessful an error message is generated, but if it is successful then there is no error message. The following definitions provide for a new type, $\mathsf{optional}\ T$, for any existing type, T, along with the predicates $\mathsf{defined}$ and $\mathsf{undefined}$ which test whether an element of $\mathsf{optional}\ T$ is defined or not. The function, the, extracts the element from a defined member of $\mathsf{optional}\ T$. We further define a prefix relation, $\mathsf{setdisjoint}$, which holds for a set of sets if all the members of that set of sets are pairwise disjoint. Lastly we define two functions that cycle forwards and backwards through non-empty injective sequences.

$$\mathsf{optional}\ [X] == \{xs : \mathbb{P}\, X \mid \# xs \leq 1\}$$

$$\begin{array}{|l}
\hline
[X] \\
\mathsf{defined}\ _, \mathsf{undefined}\ _ : \mathbb{P}(\mathsf{optional}\ [X]) \\
\mathsf{the}: \mathsf{optional}\ [X] \pfun X \\
\hline
\forall xs : \mathsf{optional}\ [X] \bullet \mathsf{defined}\ xs \Leftrightarrow \# xs = 1 \land \\
\qquad\qquad \mathsf{undefined}\ xs \Leftrightarrow \# xs = 0 \\
\forall xs : \mathsf{optional}\ [X] \mid \mathsf{defined}\ xs \bullet \\
\qquad\qquad \mathsf{the}\ xs = (\mu\, x : X \mid x \in xs) \\
\hline
\end{array}$$

$[X]$

$$\mathsf{setdisjoint}\ \_ : \mathbb{P}(\mathbb{P}(\mathbb{P}\, X))$$

$$\forall\, xss : \mathbb{P}(\mathbb{P}\, X) \bullet \mathsf{setdisjoint}\ \ xss \Leftrightarrow (\forall\, xs, ys : \mathbb{P}\, X \bullet ((xs \in xss) \land (ys \in xss) \land (xs \neq ys)) \Rightarrow (xs \cap ys) = \varnothing)$$

$[X]$

$$\mathsf{CycleNext},\ \mathsf{CyclePrevious}\colon (X \times \mathsf{iseq}\ X) \pfun X$$
$$\mathsf{Index}\colon (X \times \mathsf{iseq}\ X) \pfun \mathbb{N}$$

$$\begin{aligned}
&\forall\, s : \mathsf{iseq}\ X;\ x : X \mid x \in (\mathsf{ran}\ s) \bullet \mathsf{Index}\ (x, s) = s^{-1} x \land \\
&\quad \mathsf{Index}\ (x, s) \neq \#s \Rightarrow \mathsf{CycleNext}\ (x, s) = s(\mathsf{Index}\ (x, s) + 1) \land \\
&\quad \mathsf{Index}\ (x, s) = \#s \Rightarrow \mathsf{CycleNext}\ (x, s) = head\ s \land \\
&\quad \mathsf{Index}\ (x, s) \neq 1 \Rightarrow \mathsf{CyclePrevious}\ (x, s) = s\ (\mathsf{Index}\ (x, s) - 1) \land \\
&\quad \mathsf{Index}\ (x, s) = 1 \Rightarrow \mathsf{CyclePrevious}\ (x, s) = last\ s
\end{aligned}$$

B Alternative notion of types for links

One problem with the specification is that the type of a link is altered as we build up our model. The alternative would have been to use a free type as follows.

$$\begin{aligned}
LINK2 ::= &\ simple\langle\langle \mathsf{LINK} \rangle\rangle \\
| &\ ref\langle\langle \mathsf{LINK} \times\ \mathsf{optional}\ [\mathsf{HIGHLIGHT}] \rangle\rangle \\
| &\ span\langle\langle \mathsf{LINK} \times\ \mathsf{optional}\ [\mathsf{HIGHLIGHT}] \times\ \mathsf{optional}\ [\mathsf{HIGHLIGHT}] \rangle\rangle
\end{aligned}$$

However, this would have led to an inappropriate and unnecessarily complicated specification. For example, the basic hypertext model of hypertext would have to be written as follows.

$HYPERTEXT2$

$$\mathsf{CONTENTS}$$
$$Links : \mathbb{P}\, LINK2$$

$$Links \subseteq (\mathsf{ran}\ simple)$$
$$first\ (|\ (simple^{-1})(|\ Links\ |)\ |) \subseteq Nodes$$