Nesting and Defoliation of Index Expressions for Information Retrieval

B.C.M. Wondergem, P. van Bommel, Th.P. van der Weide

Computing Science Institute/

CSI-R9817 July 1998

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

Nesting and Defoliation of Index Expressions for Information Retrieval

B.C.M. Wondergem, P. van Bommel, and Th.P. van der Weide
Computing Science Institute, University of Nijmegen
Toernooiveld 1, NL-6525 ED, Nijmegen, The Netherlands
E-mail: bernd@cs.kun.nl

Technical Report CSI-R9817

Keywords: Index Expressions, Defoliation, Information Retrieval, Nesting.

Abstract

In this article, a formalisation of index expressions is presented. Index expressions are more expressive than keywords while maintaining a comprehensible complexity. Index expressions are well-known in Information Retrieval (IR), where they are used for characterising document contents, formulation of user interests, and matching mechanisms. In addition, index expressions have found both practical and theoretical applicability in 2-level hypermedia systems for IR. In these applications, properties of (the structure of) index expressions are heavily relied upon. However, a correct mathematical formalisation of index expressions and their properties still lacks. Our formalism is based on the structural notation of index expressions. Defoliation, which plays an important role in defining properties of index expressions, is provided as a recursively defined operator. Finally, two other representational formalisms for index expressions are compared to ours.

Contents

1 Introduction

Index expressions were defined by Bruza (see [Bru93] and [Bru90]) as structured descriptor language for Information Retrieval (IR) (see [Rij90]). Index expressions are constructed from terms (e.g. keywords, concept names, or denotations of attribute values) and connectors, representing relations between terms in the form of prepositions and gerunds. The language of index expressions is thus more powerful than that of keywords. In addition, index expressions form reasonable approximations of noun-phrases, which are seen as basic units of thought ([Win83] and [Cra78]).

The structured nature of index expressions has allowed many applications in Information Retrieval. Index expressions are, for instance, used for indexing documents and as query formulation language. Characterising document contents are expressed in terms of index expressions allows the construction of navigational overview structures.

These navigational overview structures are called lithoids ([Bru90]) and are constructed from an initial set of index expressions by exploiting the availability of sub-index expressions. Lithoids can be seen as graphs. The nodes of lithoids consist of the initial set of index expressions plus all their subexpressions. The edges connect index expressions with their subexpressions. Lithoids can be exploited for navigational query formulation.

Formulating a query in a lithoid starts at some node (index expression) in the lithoid. The user then navigates through the lithoid by selecting links that lead to different index expressions. Query formulation ends when a node is arrived at that properly describes the user's information need. Mechanisms for navigational support in lithoids and other overview structures are described in [Ber98] and [BB97].

Index expressions can also be used to obtain stratified hypermedia representations of information systems. The lithoids that were constructed from index expressions can serve as hyperindex in 2-level hypermedia representations (see e.g. [BW92] and [BW90]). An IR system that supports hyperindex navigation is described in [IWW+95].

Combination structures for lithoids, called association indices (see [WBW98]), are also defined using properties of index expressions and relations between them. Association indices serve as topmost layer in 3-level hypermedia architectures. This architecture, which strongly exploits the structured nature of index expressions, is exploited for mediated IR.

Other practical use of index expressions within IR is for matching user interests with document contents. Matching can be done with, for instance, belief networks (see [BI94] and [BG94]). An implemented retrieval engine for the disclosure of a slides library is described in [BBB91].

Finally, index expressions are also studied within theoretical IR contexts. Preferential Models for IR, capable of reasoning with user preferences, are described in [BvL97] and [Won96]. Navigation in lithoids delivers information from which a Preferential Model is constructed. Matching then is a process of inference in the constructed model. Based on the inferences, the matching can

be explained to and personalised by the user, as described in [WBHW98].

The mentioned usages of index expressions hinge on properties of index expressions. In particular, relations on index expressions are often exploited. For instance, the construction of lithoids and association indices makes heavy use of the direct subexpression relation. However, a correct and complete mathematical formalisation of subexpressions and their defoliation has not yet been given.

This lack is lifted in this article. We provide a representational formalism for index expressions as well as sound definitions for several important relations on index expressions. Furthermore, a number of properties of index expressions and relations on them are stated.

The structure of this article is as follows. Section 2 formally defines (the language of) index expressions and their defoliation. A number of subexpression relations and their properties are provided in section 3. Section 4 provides two other representational formalisms and compares these to the structural notation. In particular, it is shown that these other formalisms generate the same language.

2 Formalising Index Expressions

In this section, the language of index expressions is defined. Section 2.1 introduces index expressions based on the nesting operator and provides a number of properties of index expressions. Section 2.2 elaborates the defoliation of index expressions, i.e., the removal of their leaves.

2.1 Index Expressions

We make an explicit distinction between the empty index expression and nonempty index expressions. There is a principal difference between descriptors that carry information and descriptors that represent the absence of information. Furthermore, the distinction eases the structural definition of index expressions as well as the defoliation of index expressions.

The empty index expression is denoted by ϵ, which is a special symbol. This means that the symbol ϵ cannot appear in the set of terms or connectors on which the index expressions are based.

Definition 2.1
Empty Index Expression

$$I = \epsilon$$

□

Nonempty index expressions can incorporate a number of subexpressions. These subexpressions are nonempty index expressions themselves, allowing a nice recursive structure. Before defining the exact nature of nonempty index expressions, we first focus on *subexpressions*, also called nested index expressions.

The *nesting operator* for index expressions which is defined below is used to construct nonempty index expressions in definition 2.3. The nesting operator provides a notational shorthand for denoting the subexpressions of index expressions. It defines a series of nested subexpressions, just like the summation operator defines a series of numbers (which are to be added), by concatenating subexpressions from a left border up to a right border.

Definition 2.2

Nesting Operator for Index Expressions *For given* left border k *and* right border l, *such that* $1 \leq k \leq l$, *let* I_i *be nonempty index expressions and* c_i *be connectors* $(k \leq i \leq l)$. *The index expression nesting operator, denoted by* $\otimes$, *is defined as follows:*

$$\otimes_{i=k}^{l} c_i(I_i) =^{\texttt{def}} \begin{cases} c_k(I_k) & k = l \\ \otimes_{i=k}^{l-1} c_i(I_i)\, c_l(I_l) & k < l \end{cases}$$

□

Note that the order in which subexpressions are defined by the nesting operator is relevant. Furthermore, note that the constructor operator only defines nonempty series of subexpressions. This is a deliberate property since nonempty index expressions may not contain the empty index expression as a subexpression. We will omit brackets if they are clear from the context.

Example 2.1

Nesting Operator *For* $k = 1$ *and* $l = 2$ *we have*

$$\otimes_{i=1}^{2} c_i(I_i) \;=\; \otimes_{i=1}^{1} c_i(I_i)\, c_2(I_2) \;=\; c_1(I_1)\, c_2(I_2)$$

An instantiation of this is obtained for $c_1 =$ by, $I_1 =$ industry, $c_2 =$ of, *and* $I_2 =$ water*:*

by (industry) of (water)

□

Nonempty index expressions consist of a head, also called the lead term, and, possibly, a number of nonempty nested index expressions.

Definition 2.3

Language of Non-empty Index Expressions *Let* T *be a set of terms and* C *be a set of connectors, such that* $T \cap C = \varnothing$, *the empty index expression* $\epsilon \notin T \cup C$, *and the* null-connector $\circ \in C$. *The language of non-empty index expressions based on* T *and* C, *denoted by* $\mathcal{L}(T, C)^+$, *is defined as the smallest superset of* T *for which*

- *if* $h \in T$ *and for some right border* $l \geq 1$ *we have* $c_1, \ldots, c_l \in C$ *and* $I_1, \ldots, I_l \in \mathcal{L}(T, C)^+$, *then also* $h \otimes_{i=1}^{l} c_i(I_i) \in \mathcal{L}(T, C)^+$.

Here, the term h is called the head *or* lead term. *If T and C are clear, we write $\mathcal{L}^+$ for short.* □

Note that terms and connectors may appear more than once in an index expression. Furthermore, note that the left border of nonempty index expressions is set to one. This is our convention for the left border. It does not cause a loss of generality since the nested subexpressions can always be renumbered easily. However, note that left borders which are not equal to one are used in the defoliation of index expressions (see case 5 of def. 2.8).

Example 2.2

Nonempty Index Expressions *Since the language of nonempty index expressions is a superset of the set of terms, i.e., $T \subseteq \mathcal{L}(T, C)$, all terms are nonempty index expressions. An example nonempty index expression is*

$$E_1 \;=\; \mathsf{industry}$$

Furthermore, using water *as head,* and *as connector, and the term* air *as nonempty subexpression, the following nonempty index expression is constructed:*

$$E_2 \;=\; h \otimes_{i=1}^{1} c_i(I_i) = hc_1(I_1) \;=\; \mathsf{water\ and\ (air)}$$

In a similar way, the following index expression is constructed:

$$E_3 \;=\; \mathsf{rural} \circ \mathsf{\ (areas)}$$

Now, using $h =$ pollution *as head, connectors $c_1 =$* by*, $c_2 =$* of*, and $c_3 =$* in*, and nonempty subexpressions $I_1 = E_1$, $I_2 = E_2$, and $I_3 = E_3$, we obtain*

$$E_4 \;=\; h \otimes_{i=1}^{3} c_i(I_i) \;=\; h\, c_1(E_1)\, c_2(E_2)\, c_3(E_3)$$

$$= \mathsf{pollution\ by\ (industry)\ of\ (water\ and\ (air))\ in\ (rural} \circ \mathsf{\ (areas))}$$

□

The language of index expressions is now defined as the language of nonempty index expressions united with the empty index expression.

Definition 2.4

Language of Index Expressions *The language of index expressions is based on a set of terms T and a set of connectors C, denoted by $\mathcal{L}(T, C)$, and is defined as*

$$\mathcal{L}(T, C) \;=\; \mathcal{L}(T, C)^+ \cup \{\epsilon\}$$

If T and C are clear, we write $\mathcal{L}$ for short. □

2.1.1 Properties of Index Expressions

In this section, basic properties of index expressions are considered. First, we consider the size of index expressions. The size of an index expression is defined as the number of terms in it. It should be remarked that multiple occurrences of terms are counted. The empty index expression has size 0 since it does not contain any terms. In the size of a nonempty index expression, the head as well as the sizes of the subexpressions are counted.

Definition 2.5

Size of an Index Expression *The expression $|I|$ denotes the size of index expression I and is defined below.*

$$| \ | : \mathcal{L} \rightarrow N$$

$$\begin{array}{lcll} |\epsilon| & = & 0 & \\ |h| & = & 1 & h \in T \\ |h \otimes_{i=1}^{l} c_i(I_i)| & = & 1 + \Sigma_{i=1}^{l} |I_i| & \end{array}$$

□

Example 2.3

Size of Index Expressions *Each term $t \in T$ has size 1. For example,* $|\mathsf{industry}| = 1$. *For* $I_2 = \mathsf{water\ and\ (air)}$, *we obtain*

$$|E_2| = |h \otimes_{i=1}^{1} c_i(I_i)| = 1 + \Sigma_{i=1}^{1} |I_i| = 1 + |\mathsf{air}| = 1 + 1 = 2$$

Similarly, $|E_3| = |\mathsf{rural} \circ \ \mathsf{areas}| = 2$.

Then,

$$|E_4| = 1 + \Sigma_{i=1}^{3} |I_i| = 1 + |E_1| + |E_2| + |E_3| = 1 + 1 + 2 + 2 = 6$$

□

Next, we consider the leaves of index expressions. A leaf is a non-empty index expression without subexpressions. Therefore, the empty index expression is no leaf. Non-empty index expressions which have subexpressions, i.e., $l \geq 1$, are also no leaves. This is reflected in the boolean function `IsLeaf`.

Definition 2.6

Leaves of Index Expressions *The predicate* `IsLeaf` *specifies if an index expressions is a leaf. Leaves coincide with terms:* $\texttt{IsLeaf}(I) \Leftrightarrow I \in T$.

□

The property `IsLeaf()` is used in the defoliation operator (see definition 2.8) to ensure that valid index expressions are returned after defoliation. Obviously, all leaves have size one:

Lemma 2.1 $\texttt{IsLeaf}(I) \Leftrightarrow |I| = 1$

2.2 Defoliation of Index Expressions

In this section, we consider the defoliation of index expressions. Defoliation, meaning the removal of leaves, is an important concept in IR. For instance, the subexpression relations, which are given in the next section, are defined in terms of defoliation. Subexpressions can be obtained from the original index expression by removing a number of leaves. Thus, computing subexpressions for constructing lithoids also involves repeated defoliations.

Furthermore, in full-text searches, defoliation is exploited to form broader search terms. An example of this is the construction of so called enlargements, as described in [WHHW96] and [IWW+95]. In addition, defoliation can be used to define a measure of distance between index expressions, which can be used in the matching process of Information Retrieval. Finally, a strict form of inference is driven by defoliation, as described in [BW91].

In defoliation, a designated leaf of an index expression is removed. If possible, it also allows the removal of the head of the index expression. This case occurs if the index expression has exactly one subexpression. In the remainder of this article, we implicitly include this case when referring to leaves.

2.2.1 Pointing Sequences

Leaves can be identified by describing a downward path from the lead term. The downward path is described by a nonempty sequence of natural numbers, the elements of which iteratively specify the subexpression in which the leaf resides. For instance, in index expression E_4, the sequence $[2, 1]$ denotes the leaf air since this term is obtained by first selecting the second subexpression, i.e., $I_2 =$ water and (air), and therein the first subexpression. Since nonempty sequences of natural numbers can identify leaves, we call them *pointing sequences.*

Definition 2.7

Nonempty Sequences of Natural Numbers *Let N denote the natural numbers, i.e., $0, 1, 2, \ldots$. Furthermore, let $\vec{N}$ denote the set of nonempty sequences over the natural numbers. We denote a sequence consisting of a single element $x \in N$ by $[x]$ and a sequence with at least two elements with $[x, xs]$ where x is the first element of the sequence and $[xs]$ denotes the nonempty tail of the sequence.* □

Example 2.4

Pointing Sequence *Figure 1 shows index expression E_4 equipped with pointing sequences for all its terms. For example, the sequence $[0]$ points to the head* pollution. *The sequence $[2, 1]$ points to the term* air, *which is a leaf. Pointing sequence $[3]$ specifies term* rural, *which is not the head nor a leaf.*

□

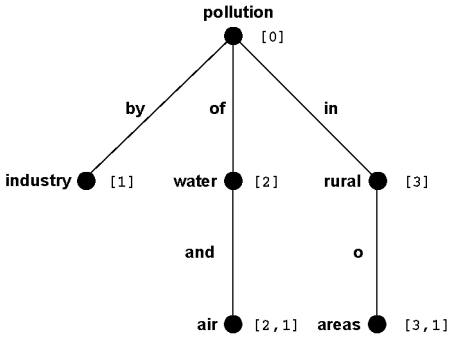

Figure 1: Example index expression with pointing sequences for all terms.

Pointing sequences and natural numbers do not add expressive power to the definitions in this article because index expressions, in particular the sizes of their subexpressions, can serve the same purpose.

Each leaf can be identified as the rightmost leaf in the subexpression that covers the part of the original index expression left of the upward path from that leaf to the head. This is illustrated in figure 2(a), where the grey area models the indicated subexpression. So, for the sake of identifying leaves, pointing sequences are not strictly necessary. They do provide, however, an easy and intuitive notation.

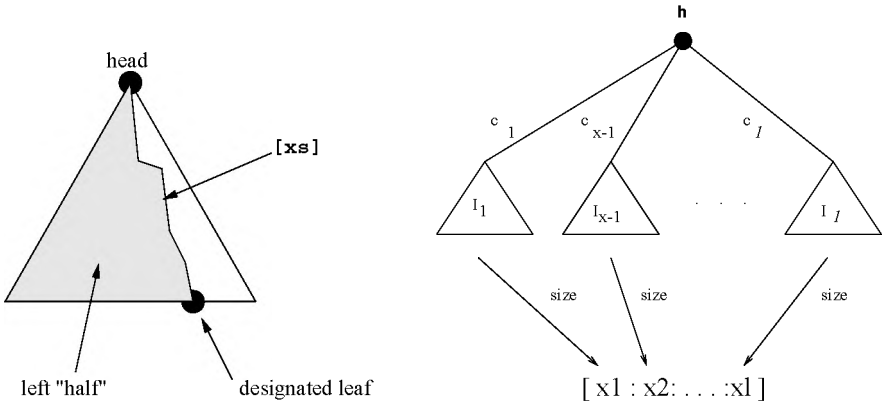

Figure 2: (a) Pointing Sequence models Index Expression. (b) Mapping Index Expressions to Pointing Sequences.

Thus, the downward path identifying a leaf can be described by an index expression I. That is, pointing sequences here serve as a simplification of index expressions. We now provide a direct mapping from index expressions to pointing sequences, showing that pointing sequences can be expressed by index expressions. In this mapping, the sizes of the subexpressions of I are mapped

to elements of a pointing sequence. The mapping is illustrated in figure 2(b). In this way, the subexpressions define the downward path in the same way as pointing sequences do, i.e., by specifying which subexpressions to select. When used for this purpose, the index expression $I = h \otimes_{i=1}^{l} c_i(I_i)$ is equivalent to the sequence $[xs]$ for which the size of subexpression I_i gives the (value of the) i-th element of $[xs]$. Note that l equals the number of elements in $[xs]$ and that the actual values of terms and connectors are of no importance.

Thus, for identifying leaves of index expressions, sequences of natural numbers do not significantly augment our theory since they can alternatively be expressed by index expressions. We can thus include (sequences of) natural numbers in our theory without relying on additional power or properties that were not catered for by index expressions themselves. The notation for nonempty sequences of natural numbers is defined below.

2.2.2 Defoliation Operator

The defoliation operator Δ removes a leaf which is indicated by a given pointing sequence. The elements of the sequence denote which subexpressions to visit in order to reach the leaf. For an index expression I and a nonempty sequence of natural numbers $[xs]$, the expression $\Delta(I, [xs])$ denotes the index expression obtained from I by removing the leaf denoted by pointing sequence $[xs]$.

Definition 2.8

Defoliation of Index Expressions *The defoliation operator Δ has an index expression and a non-empty sequence of natural numbers as arguments and delivers an index expression:*

$$\Delta : \mathcal{L} \times \vec{N} \mapsto \mathcal{L}$$

The definition of the defoliation operator is given by equations (1)..(5) below by examining the different cases that may occur. □

The defoliation operator is inductively defined, using the structure of index expressions. This leads to four cases, one of which is recursive. The three non-recursive cases are handled first. The first non-recursive case consists of defoliating an index expression that consists of a head $h \in T$ only, and therefore is a leaf. The only position a term can be defoliated at is 0, which identifies the term itself. Removing its only leaf, the index expression becomes empty:

$$\Delta(h, [0]) = \epsilon \tag{1}$$

The second non-recursive case deals with an index expression that has exactly one subexpression. If the head of this index expression is removed, by defoliating it at position 0, only the subexpression remains:

$$\Delta(hc_1(I_1), [0]) = I_1 \tag{2}$$

The last non-recursive case handles non-trivial applications of defoliation. These applications remove the non-head leaf which comprises subexpression I_x. This results in the conditions $l \geq 1$, saying there is at least one subexpression, $1 \leq x \leq l$, meaning that an existing subexpression is selected, and $\texttt{IsLeaf}(I_x)$, ensuring that a single leaf is removed. To obtain the resulting index expression, a copy is made of the original one without subexpression I_x:

$$\Delta(h \otimes_{i=1}^{l} c_i(I_i), [x]) = h \otimes_{i=1}^{l, i \neq x} c_i(I_i) \tag{3}$$

Example 2.5

Non-recursive Cases of Defoliation *This example illustrates cases (1), (2), and (3), respectively.*

Defoliating a term results in the empty index expression:

$$\Delta(\mathsf{industry}, [0]) = \epsilon$$

When an index expression with only one subexpression is defoliated at the head, the subexpression remains. For instance,

$$\Delta(\mathsf{rural} \circ \ (\mathsf{areas}), [0]) = \mathsf{areas}$$

Subexpressions that consist of a single term, i.e., leafs, can be deleted by defoliation. For instance,

$$\begin{array}{rcl} \Delta(\mathsf{water\ and\ (air)}, [1]) & = & \mathsf{water} \\ \Delta(\mathsf{pollution\ of\ (water)\ by\ (industry)}, [1]) & = & \mathsf{pollution\ by\ (industry)} \\ \Delta(\mathsf{pollution\ of\ (water)\ by\ (industry)}, [2]) & = & \mathsf{pollution\ of\ (water)} \end{array}$$

Consider index expression E_4 from example 2.2:

$$\Delta(E_4, [1]) \ = \ \mathsf{pollution\ of\ (water\ and\ (air))\ in\ (rural} \circ \mathsf{(areas))}$$

□

The recursive case for defoliation is first illustrated in example 2.6. Then, the definition is followed by an example providing a number of practical illustrations of the recursive case of defoliation.

Example 2.6

Recursive Defoliation (Schematic)

Figure 3 gives a schematic description of the recursive case of defoliation. The defoliation $\Delta(h \otimes_{i=1}^{l} c_i(I_i), [x, xs])$ is to remove the designated leaf, specified by pointing sequence $[x, xs]$, in index expression $h \otimes_{i=1}^{l} c_i(I_i)$. The designated leaf resides in subexpression I_x since x is the first element of the pointing sequence. The designated leaf is denoted locally in I_x by the remaining pointing sequence $[xs]$.

The resulting index expression has the same head h as the original index expression. Furthermore, the subexpressions left ($I_1 \ldots I_{x-1}$) and right ($I_{x+1} \ldots I_l$) of I_x are not altered. They are shown in figure 3 by the left and right triangles. Subtree I_x, depicted by the large triangle, hosts the rest of the defoliation. The call that effectuates this, $\Delta(I_x, [xs])$, will recursively remove the designated leaf. In this process, a downward path in I_x is followed, as shown by the line in I_x. This line depicts other recursive calls and one, i.e., the last, non-recursive call that actually removes the designated leaf.

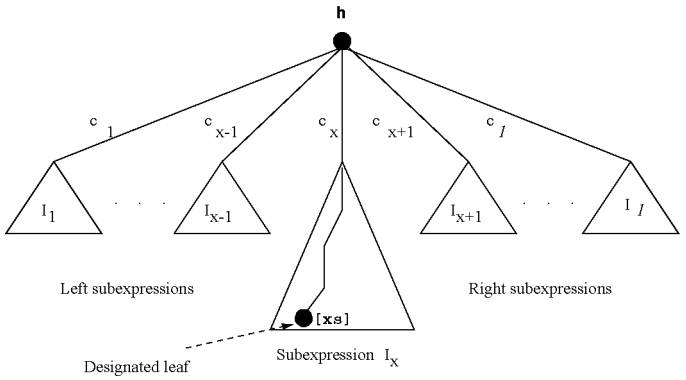

Figure 3: Schematic description of recursive defoliation.

□

Now, the definition of the recursive case of defoliation is elaborated on. Consider index expression $h \otimes_{i=1}^{l} c_i(I_i)$ and pointing sequence $[x, xs]$ which identifies the leaf to be removed. A number of conditions are required for the defoliation $\Delta(h \otimes_{i=1}^{l} c_i(I_i), [x, xs])$ to be correctly defined. The index expression should at least contain one subexpression: $l \geq 1$. Furthermore, the defoliation must take place in an existing subexpression. Therefore, the first element x of the pointing sequence should fall between the left and right border: $1 \leq x \leq l$. Finally, as the selected subexpression I_x may not become empty after defoliation, I_x may not consist of a single leaf: `not IsLeaf`(I_x).

The defoliated index expression is obtained as follows. First, all subexpressions left of the selected subexpression I_x ($\otimes_{i=1}^{x-1} c_i(I_i)$) are copied. Connector c_x, that connects the head to the selected subexpression, is copied as well. Then, defoliation proceeds by the recursive call $\Delta(I_x, [xs])$ which removes the leaf that is denoted relatively in subexpression I_x by $[xs]$. The result of the recursive call $\Delta(I_x, [xs])$ is inserted in the correct position, i.e., after connector c_x. Finally, the subexpressions right of I_x ($\otimes_{i=x+1}^{l} c_i(I_i)$) are copied:

$$\Delta(h \otimes_{i=1}^{l} c_i(I_i), [x, xs]) =^{\mathtt{def}} h \underbrace{\otimes_{i=1}^{x-1} c_i(I_i)}_{\text{left subs}} c_x(\Delta(I_x, [xs])) \underbrace{\otimes_{i=x+1}^{l} c_i(I_i)}_{\text{right subs}} \quad (4)$$

If I_x were to be a leaf, the recursive application of the defoliation operator could later result in leaving an empty descriptor ϵ in the index expression, which is not allowed in valid index expressions (see definition 2.4). The $\mathtt{notIsLeaf}(I_x)$ is thus necessary to prevent the use of ϵ as subexpression. This is guaranteed by preventing a recursively initiated application of $\Delta(h, [0])$. For example, without this last condition, $\Delta(hc_1(t_1), [1, 0])$ would lead via $hc_1(\Delta(t_1, [0]))$ to $hc_1(\epsilon)$.

The recursive case of the defoliation operator is stated as a generic case for four subcases. Special instantiations of case 4 occur if there is only one subexpression or if defoliation is to proceed in the first or last subexpression. This is because the nesting constructor $\otimes$ always renders a non-empty result. In the case defoliation is to proceed in the first subexpression, the part $\otimes_{i=1}^{x-1} c_i(I_i)$ is skipped. In the case of proceeding in the last subexpression, the part $\otimes_{i=x+1}^{l} c_i(I_i)$ is left out. In the case of a single subexpression, both parts are left out.

Example 2.7

Applications of Recursive Defoliation *Again, consider* $E_4 =$ pollution by (industry) of (water and (air)) in (rural $\circ$ (areas)). *Suppose we want to remove the leaf* air *from* E_4. *This leaf is denoted by pointing sequence* $[2, 1]$ *(see also figure 1).*

The first call,

$$\Delta(\text{pollution by (industry) of (water and (air)) in (rural} \circ \text{(areas))}, [2, 1])$$

is covered by the recursive case. The recursive call that results from applying this case, $\Delta(I_2, [1])$, *proceeds in subexpression* $I_2 =$ water and (air). *In* I_2, *leaf* air *is denoted by the pointing sequence* $[1]$. *The left subexpressions, here only* $I_1 =$ industry, *and the right, here* $I_3 =$ rural $\circ$ (areas) *are not altered. Thus, after this first step we have*

$$\text{pollution by (industry) of}(\Delta(\text{water and (air)}, [1])) \text{ in (rural} \circ \text{ (areas))}$$

The second application of the defoliation operator, $\Delta(\text{water and (air)}, [1])$, *was illustrated in example 2.5 that showed that the result of this is the single term* water. *The final result therefore becomes*

$$\text{pollution by (industry) of (water) in (rural} \circ \text{ (areas))}$$

□

All other cases of defoliation explicitly render the value undefined:

$$\Delta(I, \vec{x}) =^{\mathtt{def}} \text{undefined}, \quad \mathtt{otherwise} \quad (5)$$

Example 2.8

Undefined cases of defoliation. *Undefined defoliations occur in three ways. First, defoliating the empty descriptor is invalid. Second, pointing sequences that go beyond the depth of the index expression cannot be used. Finally, selecting a subexpression that is* out of range *is also invalid.*

First of all, the empty descriptor cannot be defoliated, since it contains no leaves. The following is thus an example of an undefined defoliation

$$\Delta(\epsilon, [1])$$

Second, pointing sequences that are too long, i.e., go beyond the depth of an index expression, cannot be used for defoliation. If the elements of the pointing sequence specify existing subexpressions, the largest prefix of the pointing sequence is processed correctly. In the end, however, defoliation can no longer proceed and, for some term $t \in T$ and pointing sequence with at least two elements $[x, xs]$ results in

$$\Delta(t, [x, xs])$$

Finally, an example of selecting a subexpression that is out of range is given below. Here, the third subexpression is selected by the first element of the pointing sequence. However, the index expression has only two subexpressions.

$$\Delta(hc1(I_1)c_2(I_2), [3, xs])$$

□

We say that $\Delta(I, \vec{x})$ is defined iff it can the computation by the above rules uses only the defined cases. Unless stated otherwise, we will only consider defined applications of Δ in the remainder of this article.

2.2.3 Properties of Defoliation

In this subsection, a number of properties of defoliation are considered. First, the fact that defoliation actually renders an index expression with one leaf less is stated. Then, two lemmas about the termination of defoliation are given.

Since the defoliation operator removes exactly one leaf and the size of a leaf is 1, the lemma below is valid.

Lemma 2.2 One Leaf Less For every nonempty index expression $I \in \mathcal{L}^+$ and every pointing sequence $\vec{x}$ such that $\Delta(I, \vec{x})$ is defined, we have $|\Delta(I, \vec{x})| = |I| - 1$.

Proof:
Suppose the defoliation is defined. Defoliation terminates in case 1, 2, or 3. In all these cases, the resulting index expression has one term less than the original index expression.

Termination of defoliation can be viewed from two points. First of all, as the termination of a single defoliation. This is an important property since otherwise defoliation is wrongly defined. This is captured in the following lemma. Second, as termination of repeated defoliation, which is covered by lemma 2.4.

Lemma 2.3 Termination of Defoliation (I) For every index expression I and pointing sequence $\vec{x}$, the computation of $\Delta(I, \vec{x})$ terminates.

Proof:
If the defoliation is undefined, case 5 is applied, after which the computation terminates.

If the defoliation is defined, case 5 does not occur. In the recursive case (4), the defoliation operator is applied to a smaller index expression in the recursive call. This means that, in the end, defoliation terminates in one of the non-recursive cases, i.e., case 1, 2, or 3.

Every index expression can be transformed to the empty index expression by defoliating it repeatedly. The number of defoliations needed is equal to the size of the index expression. This property ensures that every index expression can ultimately be broken down to the empty index expression. It shows that the empty descriptor appears in every lithoid, since it is a subexpression of every index expression.

Lemma 2.4 Termination of Defoliation (II) For every non-empty index expression $I \in \mathcal{L}^+$ there exist vectors $\vec{x_1}, \ldots, \vec{x_n} \in \vec{\mathrm{N}}$ such that $n = |I|$ and $\Delta(\Delta(\ldots(\Delta(I, \vec{x_1}), \vec{x_2}) \ldots \vec{x_n}) = \epsilon$.

Proof:
For every nonempty index expression I, there is a pointing sequence $\vec{x}$ such that $\Delta(I, \vec{x})$ is defined and, by lemma 2.2, contains one term less than I. Defoliation can thus be repeated exactly $|I|$ times until the empty index expression is obtained.

Note that, in the last lemma, $\vec{x_j}$ is dependent on $\vec{x_i}$ $(1 \leq i < j)$ since the $\vec{x_i}$ sequences modify the structure of the index expression that is defoliated.

3 Subexpressions

Subexpressions of index expressions play an important role in many applications of index expressions in IR. For instance, the construction of lithoids and association indices is based on properties of subexpressions. This also holds for navigational actions in these structures. In addition, the mapping of, for instance, navigational behaviour to formal models hinges on properties of the notion of subexpressions. Finally, subexpressions allow the definition of matching functions.

In the following subsections, three subexpression relations are given. In each subsection, the subexpression relation at hand is defined followed by a lemma stating its properties. The subexpression relations have different properties which makes them suitable for different applications. In matching, for instance, the general subexpression has to be used if exact matches are to be found. Then, proper applications and additional properties are stated.

3.1 Direct Subexpressions

The defoliation operator is now used to construct the direct subexpression relation $\prec_d \subseteq \mathcal{L}^2$ for index expressions. Direct subexpressions of index expressions are obtained by removing exactly one leaf.

Definition 3.1
Direct Subexpression Relation

$$I \prec_d J \stackrel{\texttt{def}}{=} \exists \vec{x} \in \vec{N} : \Delta(J, \vec{x}) = I$$

□

The following lemma holds for the direct subexpression relation:

Lemma 3.1 $\prec_d$ **is irreflexive and asymmetric**

Proof:
Suppose there is some index expression $I \in \mathcal{L}$ for which $I \prec_d I$. This would mean that $\exists \vec{x} \in \vec{\mathrm{N}} : \Delta(I, \vec{x}) = I$. Using lemma 2.2, this would result in $|\Delta(I, \vec{x})| = |I| - 1$, leading to a contradiction.
Suppose $\prec_d$ is not asymmetric, that is, $\texttt{not}(I \prec_d J \Rightarrow J \not\prec_d I)$. Let I and J be such that $I \prec_d J$ and $J \prec_d I$. Then, applying lemma 2.2 results in $|I| = |J| - 1$ and $|J| = |I| - 1$, which is a contradiction.

The fact that direct subexpressions differ exactly one leaf, allows fine-grained navigation in QBN. So called one-step refinements, as proposed in [Won96], coincide with instances of the direct subexpression relation. In QBN, the set of choices at each focus then consists of one-step refinements and enlargement since the edges in lithoids coincide with the direct subexpression relation (see [BW92]). This property is exploited in so called hyperindex browsers (for an example, see [IWW+95]) which allow step-wise reformulation of a user query based on intermediately retrieved documents.

Search paths that are constructed during navigation in lithoids can nicely be given semantics. An example of this is described in [Won96], where search paths impose a ranked clustering of the documents underlying the lithoid. Another example, as described in [BvL97] and [WHHW96], constructs Preferential Models out of search paths. Preferential Models, which are capable of capturing (non-monotonic) user preferences, are augmented with domain knowledge in [WBHW98].

In addition, the direct subexpression relation serves as the basis for defining the edges of more complicated navigational structures such as association indices ([WBW98]). The fine-grained nature of the subexpressions are exploited here as well. The navigational actions that can be performed in association indices are also defined on the basis of the direct subexpression relation.

The following properties are now obvious:

Lemma 3.2 Additional Properties of Direct Subexpressions

1. For each index expression I and nonempty pointing sequence $\vec{x}$ such that $\Delta(I, \vec{x})$ is defined, we have $\Delta(I, \vec{x}) \prec_d I$.
2. There is no I for which $I \prec_d \epsilon$.
3. $I \prec_d J \Rightarrow |J| = 1 + |I|$
4. $I_1 \prec_d J$ **and** $I_2 \prec_d J \Rightarrow |I_1| = |I_2|$

Proof:
The first property follows directly from definition 3.1.
The second property follows from definition 3.1 and the observation that the empty index expression cannot be defoliated.
The third property is a consequence of the combination of definition 3.1 and lemma 2.2. Note that the property does not hold the other way around.
The last property follows from applying the previous property to both $I_1 \prec_d J$ and $I_2 \prec_d J$: $|I_1| + 1 = |J| = |I_2| + 1$. Again, note that this property does not hold the other way around.

The first property illustrates that defoliation defines direct subexpressions. The second property guarantees that ϵ is the bottom of navigation structures that are based on the direct subexpression relation. The third and fourth property are direct consequences of our defoliation-based definition of the direct subexpression relation. These properties are used in, for instance, proving properties of association indices ([WBW98]).

3.2 Strict Subexpressions

The strict subexpression relation for index expressions $\prec \subseteq \mathcal{L}^2$ is defined as the transitive closure of the direct subexpression relation. The transitive closure of a relation R is denoted by R^*.

Definition 3.2
Strict Subexpression Relation

$$\prec =^{\mathtt{def}} \prec_d^*$$

□

The following lemma is now clear:

Lemma 3.3 $\prec$ **is irreflexive, asymmetric, and transitive**

Proof:
Irreflexivity, i.e. $I \not\prec I$, and asymmetry, i.e. $I \prec J \Rightarrow J \not\prec I$, follow directly from lemma 3.1 and definition 3.2: only transitive instances are added to $\prec_d$.
Transitivity ($I \prec J$ **and** $J \prec K \Rightarrow I \prec K$) follows from definition 3.2: $\prec$ is the transitive closure of $\prec_d$.

Transitivity and irreflexivity of $\prec$ imply that the strict subexpression relation is a quasi order on $\mathcal{L}$.

The strict subexpression relation can be exploited for personalising search paths in QBN ([Ber98]). The reason for this is that it not only includes one-step refinements and enlargments. In particular, the source and destination of search paths that consist of repeated refinements or enlargement belong to the strict subexpression relation. Therefore, these search paths can be replaced by a single pair of index expressions. This means that, when this pair is offered to the user as an extra option, shortcuts are created making navigation more efficient.

Lemma 3.4 Additional Properties of Strict Subexpression Relation

1. $\epsilon \prec I$ for every non-empty I
2. $\prec_d \subseteq \prec$

Proof:
The first property follows from the observation that every non-empty index expression can be defoliated to the empty descriptor (lemma 2.4) and transitivity of $\prec$.

The second property follows directly from definition 3.2: $\prec$ is a superset of $\prec_d$.

The first property can be exploited by taking ϵ as a starting point in QBN. The fact that the user has not specified any aspect of his information need yet is captured by the non-informational empty descriptor.

The second property shows that the strict subexpression relation can be used, for instance, to augment the navigational choices in QBN. This approach, as described above, provides shortcuts in $\prec$ of often chosen paths of consecutive choices within $\prec_d$.

3.3 General Subexpressions

The third relation on index expressions defined in this article is the (general) subexpression relation $\preccurlyeq \subseteq \mathcal{L}^2$ for index expressions. An index expression I is a (general) subexpression of another index expression J iff I is a strict subexpression of or equal to J. Therefore, the general subexpression relation contains the strict subexpression relation and reflexive tuples of index expressions.

Definition 3.3
Subexpression Relation

$$\preccurlyeq =^{\texttt{def}} \prec \cup \{(I, I) | I \in \mathcal{L}\}$$

□

The following lemma holds for general subexpressions:

Lemma 3.5 $\preccurlyeq$ **is reflexive, antisymmetric, and transitive**

Proof:
Reflexivity follows from definition 3.3 and lemma 3.3.
To prove antisymmetry ($I \preccurlyeq J$ `and` $J \preccurlyeq I \Rightarrow I = J$), assume $I \preccurlyeq J$ and $J \preccurlyeq I$. Assume $I \neq J$ for the sake of contradiction. Now, $I \preccurlyeq J$ and $I \neq J$ imply $I \prec J$. This means $J \not\prec I$ (by lemma 3.3) which, again with $I \neq J$, forbids $J \preccurlyeq I$. Thus, $I = J$.
Transitivity follows from definition 3.3 and lemma 3.3.

In other words, $\preccurlyeq$ is a partial order for the language of index expressions. This means that $\langle \mathcal{L}, \preccurlyeq \rangle$ is a poset. In [Won96], $\langle \mathcal{L}, \preccurlyeq \rangle$ is therefore called a structured descriptor language.

The general subexpression relation can be used to further augment the possible choices in QBN. The reflexiveness of the subexpression relation guarantees that the set of choices constructed from a certain focus include that focus itself. This is necessary for some enhanced navigational query formulation mechanisms, such as Berry Picking ([Bat89]).

In addition, the general subexpression relation can be used to define the set of relevant documents with respect to an index expression query. Every document that contains an index expression such that the query is a general subexpression of that, is deemed relevant. This includes documents that exactly match the query and documents that contain more specific index expressions.

We have defined three relations that consider the structural composition of index expressions and illustrated their possible use. We obtained the direct subexpression relation by using defoliation, and then derived the other two relations. Other approaches may first obtain one of these other two relations. This is only a practical difference, however, since from any of the three relations, the other two can be derived. For each application or use, the most suitable relation can be chosen.

4 Other Representations for Index Expressions

In section 2.1, we introduced the structural notation of index expressions, which is based on the nesting operator. This section shows two additional representation formalisms for index expressions: the grammar representation and the broaden-based representation. It is shown that these formalisms generate exactly the same language of index expressions $\mathcal{L}(T, C)$ from definition 2.4.

4.1 Grammar Representation

As stated in the introduction, one of the uses of index expressions in IR systems is for characterising documents. This means that the document content has to be parsed to obtain index expressions. Parsers exploit *grammars* that describe the structure of the language to be parsed. Therefore, many IR systems exploit a grammar for index expressions. Definition 4.1 provides a grammar representation for index expressions. This grammar can also be used to derive the semantics of index expressions.

Definition 4.1

Grammar Representation of Index Expressions *Given sets T of terms and C of connectors as before, the language of index expressions can be described by the following grammar which is denoted in extended BNF format:*

$$
\begin{array}{lcl}
\mathsf{Expr} & \rightarrow & \epsilon \mid \mathsf{NExpr} \\
\mathsf{NExpr} & \rightarrow & \mathsf{Term}\ \{\mathsf{Connector}\ (\mathsf{NExpr})\}^* \\
\mathsf{Term} & \rightarrow & \mathsf{t},\ \mathsf{t} \in T \\
\mathsf{Connector} & \rightarrow & \mathsf{c},\ \mathsf{c} \in C
\end{array}
$$

□

The grammar representation is also called the abstract syntax representation. In this definition, Expr stands for index expression and NExpr for non-empty index expression.

Theorem 4.1 Grammar Representation ≡ Structural Representation
The grammar representation for index expressions generates the same language as the structural representation of index expressions.

Proof:

1. Every index expression generated by the grammar representation can also be described by the structural representation.

   Take a random well-formed parse tree from the abstract syntax for index expressions. At this moment, there are two possibilities. First, the parse tree is Expr $\rightarrow \epsilon$ describing the empty index expression ϵ which is also in the structural representation.

   Second, the parse tree starts with Expr $\rightarrow$ NExpr. This case is treated with induction to the number of the NExpr nonterminal in the parse-tree.

   Basis step: A single occurrence. In this case, the parse tree is Expr $\rightarrow$ NExpr $\rightarrow$ Term $\rightarrow$ t, where t is a term. Terms belong to the language of index expressions.

The induction hypothesis is that all parse trees starting with Expr $\rightarrow$ NExpr which contain at most n occurrences of the NExpr nonterminal generate an index expression that is also part of the structural representation.

Induction step: consider a parse tree with $n+1$ occurrences of the NExpr nonterminal. This parse tree starts with Expr $\rightarrow$ NExpr $\rightarrow$ Term $\{$Connector NExpr$\}^l$ for some $l \geq 0$. By the induction hypothesis, all NExpr$_i$ $(1 \leq i \leq l)$ are the root of a parse tree that contains at most n occurrences of the NExpr non-terminal and thus yield valid index expressions I_i. These are covered by the nested index expressions I_i in index expression $I = h \otimes_{i=1}^{l} c_i(I_i)$ which corresponds to the complete parse tree. In I, $h \in T$ corresponds to non-terminal Term, which is covered by the basis step of the induction, and the Connector non-terminals, which are mapped to connectors in the fourth rule of the grammar, are covered by the $c_i \in C$ of the structural representation.

2. Every index expression generated by the structural representation is also generated by the abstract syntax.

   Take a random index expression I from the language $\mathcal{L}$. We prove this case with induction to the number of occurrences of the nesting operator $\otimes$ in I.

   Basis step: No occurrences of the nesting operator. This case gives two possibilities. The first one, $I = \epsilon$, is covered by the parse tree consisting of Expr $\rightarrow \epsilon$. The second case, $I = t$, for $t \in T$, is covered by the parse tree consisting of Expr $\rightarrow$ NExpr $\rightarrow$ Term $\rightarrow t$.

   The induction hypothesis is that all index expressions that contain at most n occurrences of the nesting operator are generated by the grammar.

   The only possibility left is $I = h \otimes_{i=1}^{l} c_i(I_i)$ where I contains $n+1$ occurrences of the nesting operator. The subexpressions I_i contain at most n occurrences of the nesting operator, and thus, by the induction hypothesis, are covered by a parse tree. Index expression I then corresponds to the parse tree that starts with Expr $\rightarrow$ NExpr $\rightarrow$ Term$\{$Connector NExpr$\}*$, where $*$ is substituted by l. Head h is generated by Term $\rightarrow h$ and connectors c_i are generated from the Connector non-terminals by the fourth rule of the grammar: Connector $\rightarrow c_i$. Putting the parse trees for subexpressions I_i under the NExpr$_i$ nonterminals completes the parse tree for I.

4.2 Broaden-based Representation

The broaden-based representation is defined in terms of the binary constructor operator `broaden` : $\mathcal{L}^+ \times C \times \mathcal{L}^+ \mapsto \mathcal{L}^+$. This operator broadens an index expression I with a connector c and an index expression J, and results in a

larger index expression $\texttt{broaden}(I, c, J)$. The broaden operator, see figure 4, is used in the parsing mechanism described in [Bru93].

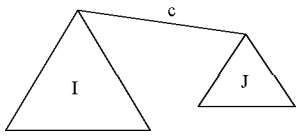

Figure 4: Broadening an index expression: $\texttt{broaden}(I, c, J)$.

An advantage of this representation lies in the fact that the constructor operator is binary in the sense that among its arguments are exactly two index expressions. The advantage of this is that induction on the structure of index expressions then only has to consider binary cases as well. At the same time, this may be seen as a disadvantage since some proofs become longer and tedious. Furthermore, the actual structure of index expressions, i.e., lead term plus a number of subexpressions, is less clearly depicted than in our formalism.

Definition 4.2

Broaden-based Non-Empty Index Expressions *The language of non-empty index expressions, denoted by $\mathcal{L}^+$, is defined as the smallest superset of T such that if I and J are non-empty index expressions and $c \in C$ is a connector, then* $\texttt{broaden}(I, c, J)$ *is also an index expression.* □

Definition 4.3

Broaden-based Language of Index Expressions

$$\mathcal{L} =^{\texttt{def}} \mathcal{L}^+ \cup \{\epsilon\}$$

□

Theorem 4.2 Broaden-based ≡ Structural Representation

Proof:

1. Every index expression generated by the broaden-based representation can also be described by the structural representation.

   This case is proven by induction to the number of occurrences of the `broaden` operator in index expressions.

   Take a random index expression I generated by the broaden-based representation. Basis step: No occurrences of the `broaden` operator. This case has two possibilities: the empty expression and single terms. For both possibilities, it is easy to see the property holds.

   The induction hypothesis is that every broaden-based index expression with at most n occurences of the `broaden` operator can also be described by the structural representation.

Induction step: assume I contains $n+1$ occurrences of the `broaden` operator. We can represent I by $\texttt{broaden}(I_1, c, I_2)$, where I_1 and I_2 are non-empty index expressions. Since I_1 and I_2 contain at most n occurrences of the `broaden` operator, they are, by the induction hypothesis, described by the structural representation. Assume their structural counterparts are J_1 and J_2, where $J_1 = h \otimes_{i=1}^{l} c_i(J_{1_i})$. Then, the I's structural counterpart is $h \otimes_{i=1}^{l} c_i(J_{1_i})cJ_2$ which is a valid structural representation of an index expression.

2. Every index expression generated by the structural representation can also be described by the broaden-based representation.

   This case is proven by induction on the number of occurrences of the nesting operator in index expressions.

   Consider an index expression I generated by the structural representation. Basis step: No occurrences of the nesting operator. This case has two possibilities: the empty index expression and single terms. For both types of index expressions, the property is directly obvious from definition 4.3.

   The induction hypothesis is that every index expression with at most n occurrences of the nesting operator can also be described by the broaden-based representation.

   Induction step: consider an index expression $I = h \otimes_{i=1}^{l} c_i(I_i)$ containing $n+1$ occurrences of the nesting operator which is generated by the structural representation. Since the subexpressions I_i contain at most n occurrences of the nesting operator, the induction hypothesis states that they can also be described by the broaden-based representation. For subexpression I_i, let J_i denote its broaden-based representation. Index expression I then is equivalent to the broaden-based representation: $\texttt{broaden}(\ldots\texttt{broaden}(\texttt{broaden}(h, c_1, J_1), c_2, J_2)\ldots, c_l, J_l)$.

Lemma 4.1 Broaden-based $\equiv$ Abstract Syntax Direct consequence of theorems 4.1 and 4.2.

5 Conclusions

In this article, we discussed issues concerning the structural representation of index expressions. This representation exploits the nesting operator for subexpressions. We provided a defoliation operator for index expressions. This operator was used to define three subexpression relations for index expressions. We indicated a number of applications for the particular subexpression relations. Several properties of index expressions and the subexpression relations were provided. Finally, our formalism was compared to two other often used representations of index expressions.

In Information Retrieval, index expressions are used for characterising documents, query formulation, and matching. Furthermore, stratified architec-

tures which allow navigational mechanisms are based on index expressions. The claims about index expressions these applications make have now been validated. This means that the many applications of index expressions that exploit their properties now have a well-defined theoretical basis. For example, lithoids and association indices can now be properly defined.

Further research can be conducted into normalisations and similarity of index expressions as aimed at in [Ber98]. In addition, the relation with noun-phrases can be further investigated. A result of such research could be an extended form of index expressions.

References

[Bat89] M.J. Bates. The design of browsing and berrypicking techniques for the on-line search interface. *Online Review*, 13(5):407–431, 1989.

[BB97] F.C. Berger and P. van Bommel. Augmenting a characterization network with semantical information. *Information Processing & Management*, 33(4):453–479, 1997.

[BBB91] R. Bosman, R. Bouwman, and P.D. Bruza. The Effectiveness of Navigable Information Disclosure Systems. In G.A.M. Kempen, editor, *Proceedings of the Informatiewetenschap 1991 conference*, Nijmegen, The Netherlands, 1991.

[Ber98] F.C. Berger. *Navigational Query Construction in a Hypertext Environment.* PhD thesis, Department of Computer Science, University of Nijmegen, September 1998.

[BG94] P.D. Bruza and L.C. van der Gaag. Index Expression Belief Networks for Information Disclosure. *International Journal of Expert Systems*, 7(2):107–138, 1994.

[BI94] P.D. Bruza and J.J. IJdens. Efficient Probabilistic Inference through Index Expression Belief Networks. In *Proceedings of the Seventh Australian Joint Conference on Artificial Intelligence (AI94)*, pages 592–599. World Scientific, 1994.

[Bru90] P.D. Bruza. Hyperindices: A Novel Aid for Searching in Hypermedia. In A. Rizk, N. Streitz, and J. Andre, editors, *Proceedings of the European Conference on Hypertext - ECHT 90*, pages 109–122, Cambridge, United Kingdom, 1990. Cambridge University Press.

[Bru93] P.D. Bruza. *Stratified Information Disclosure: A Synthesis between Information Retrieval and Hypermedia.* PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[BvL97] P.D. Bruza and B. van Linder. Preferential Models of Refinement Paths. In *Proceedings of the IJCAI-97 Workshop on AI and Digital Libraries*, 1997.

[BW90] P.D. Bruza and Th.P. van der Weide. Two Level Hypermedia - An Improved Architecture for Hypertext. In A.M. Tjoa and R. Wagner, editors, *Proceedings of the Data Base and Expert System Applications Conference (DEXA 90)*, pages 76–83, Vienna, Austria, 1990. Springer-Verlag.

[BW91] P.D. Bruza and Th.P. van der Weide. The Modelling and Retrieval of Documents using Index Expressions. *ACM SIGIR FORUM (Refereed Section)*, 25(2), 1991.

[BW92] P.D. Bruza and Th.P. van der Weide. Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal*, 35(3):208–220, 1992.

[Cra78] T. Craven. Linked phrase indexing. *Information Processing & Management*, 14(6):469–476, 1978.

[IWW+95] R. Iannella, N. Ward, A. Wood, H. Sue, and P. Bruza. The open information locator project. Technical report, Resource Discovery Unit, Resource Data Network, Cooperative Research Centre, University of Queensland, Brisbane, Australia, 1995.

[Rij90] C.J. van Rijsbergen. *Information Retrieval.* Butterworths, London, United Kingdom, 1990.

[WBHW98] B.C.M. Wondergem, P. van Bommel, T.W.C. Huibers, and Th.P. van der Weide. Domain Knowledge in Preferential Models. In Janis Barzdins, editor, *Proceedings of the third Baltic Workshop DB&IS98*, volume 1, pages 126 – 138, Riga, Latvia, April 1998.

[WBW98] B.C.M. Wondergem, P. van Bommel, and Th.P. van der Weide. The Association Index Architecture for Information Brokers. Technical report, University of Nijmegen, Nijmegen, The Netherlands, July 1998. To appear.

[WHHW96] B.C.M. Wondergem, W. van der Hoek, T.W.C. Huibers, and C. Witteveen. Preferential Semantics for Query by Navigation. In K. van der Meer, editor, *Informatiewetenschap 1996*, pages 153–168, Delft, the Netherlands, December 1996.

[Win83] W. Winograd. *Language as a Cognitive Process.* Addison-Wesley Pub. Co., Reading MA, USA, 1983.

[Won96] B.C.M. Wondergem. Preferential Structures for Information Retrieval. Master's Thesis INF-SCR-96-21, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, August 1996.