Knowledge and Information Systems (2000) 2: 33–52
© 2000 Springer-Verlag London Ltd.

Knowledge and Information Systems

Nesting and Defoliation of Index Expressions for Information Retrieval

Bernd Wondergem, Patrick van Bommel, Theo van der Weide
Computing Science Institute, University of Nijmegen, The Netherlands

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, the presupposed mathematical formalisation of index expressions and their properties still lacks. Our formalism is based on the structural notation of index expressions. It is complete in the sense that several notions of subexpressions and defoliation of index expressions are also formalised. 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.

Keywords: Defoliation; Index Expressions; Information Retrieval; Nesting

1. Introduction

Index expressions were defined by Bruza (1990, 1993) as structured descriptor language for Information Retrieval (IR) (Rijsbergen, 1979). 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 (Craven, 1978; Winograd, 1983).

Index expressions capture some of the linguistic structure of natural languages.

Received 9 Jul 1998
Revised 20 Apr 1999
Accepted 28 May 1999

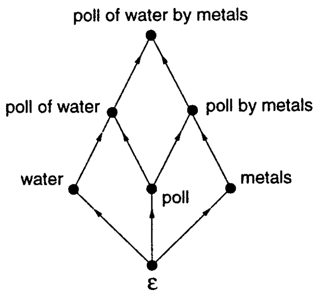

Fig. 1. Example lithoid.

In addition, the semantics of index expressions follows readily from their structure, which, however, is not too complex to harm fast parsing. Index expressions have found many applications in IR which vary from IR fundamentals as document indexing, query formulation, and matching, to the construction and usage of several stratified hypermedia models including corresponding logical models. Section 2 elaborates on applications of index expressions in IR.

The mentioned usages of index expressions hinge on properties of index expressions. In particular, subexpression relations on index expressions are often exploited. For instance, the construction of the stratified hypermedia models makes heavy use of the direct subexpression relation. In addition, the logical applications presuppose a clear and detailed definition of the syntax and semantics of index expressions. However, a correct and complete mathematical formalisation of index expressions and their defoliation (subexpressions) 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 illustrates the many usages of index expressions in Information Retrieval as well as the need for a sound formalisation. Section 3 formally defines (the language of) index expressions and their defoliation. A number of subexpression relations and their properties are provided in Section 4. Section 5 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. Index Expressions in Information Retrieval

This section provides an overview of applications of index expressions in Information Retrieval.

Characterising document contents with index expressions allows the construction of navigational overview structures. These navigational overview structures are called *lithoids* (Bruza, 1990) because of their crystalline structure. Lithoids can be seen as graphs as illustrated in Fig. 1 which shows the lithoid for pollution of water by metals. The nodes of the lithoid consist of the initial index expression plus all its *subexpressions*. The expression ϵ denotes the empty index expression. The edges connect nodes with their *direct subexpressions*. Combining lithoids for several index expressions by node sharing allows the construction of navigational overview structures for collections of documents. Although the notions of (direct) subexpressions may be intuitively clear, they have not been defined properly yet.

Applications of index expressions that are based on lithoids, and thus hinge on subexpression relations, suffer from the same problem. A number of ways in which lithoids are exploited is presented next.

Lithoids that are constructed from document characterisations serve as hyperindex in 2-level hypermedia representations (see e.g. Bruza and van der Weide, 1990, 1992). In this way, stratified hypermedia representations of information retrieval systems are obtained.

Lithoids support navigational query formulation by a process coined *Query by Navigation* (QBN) (see e.g. Bruza and van der Weide, 1990). In QBN, one starts at some node (index expression) in the lithoid and then navigates through the lithoid by iteratively selecting an adjacent link that leads to a different index expressions. In each step, a *refinement* or an *enlargement* can be selected. QBN ends when a node is arrived at that properly describes the user's information need. Refinements coincide with direct superexpressions and enlargements with direct subexpressions. This again illustrates the need for a proper definition of subexpressions. In addition, the design of algorithms for computing refinements and enlargements is best based on a thorough analysis of these phenomena. The defoliation operator, as provided in Section 3.2, may be exploited for constructing such as algorithm for enlargements.

QBN has been implemented in a number of information systems. As IR system that supports QBN on the Internet is described in Iannella et al (1995). An implemented retrieval engine based on QBN for the disclosure of a slides library is described in Bosman et al (1991). Furthermore, QBN still receives attention in ongoing research. For instance, mechanisms for navigational support during QBN are described in Berger et al (1996), Berger and van Bommel (1997) and Berger (1998). The support, which is based on a probabilistic model of QBN search paths, aims at estimating the final node and helps the user by suggesting short cuts.

Logical frameworks for QBN search paths have been developed. Preferential Models for IR, capable of reasoning with user preferences, are described in Wondergem (1996) and Bruza and van Linder (1997). QBN results in a search path, i.e. a series of user preferences, from which a Preferential Model is constructed. Matching in Preferential Models is a process of inference. Which inference rules are valid in Preferential Models depends on properties of the user preferences. Since these are specified in terms of index expressions, the availability of properties of index expressions is important. Based on the inferences, the matching can be explained to and personalised by the user, as described in Wondergem et al (1998a).

In addition, a form of strict inference based on enlargements is provided in Bruza and van der Weide (1991). An implementation of these inference is facilitated if procedures for computing enlargements are made available. Again, such procedures may be readily devised based on our defoliation operator.

Combination structures for lithoids, called *association indices* (see Wondergem et al, 1998b,c), are also defined using properties of index expressions and relations between them. Association indices serve as topmost layer in a 3-level hypermedia architecture, coined the *Association Index Architecture* (AIA). The lower levels consist of a combined lithoid for the document base and a combined lithoid that describes user interests. The AIA, 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 Bruza and Ijdens, 1994; Bruza and van der Gaag, 1994). To construct belief networks for index expressions, it is necessary that index expressions can be broken down into their constituents. The arrows in belief networks are instances of the direct subexpression relation. In addition, matching can be done by (numerical) similarity functions on index expressions. In Wondergem et al (2000) several such similarity measures are provided which preserve properties of index expressions. Two of these properties are embedding, a non-contiguous notion of subexpressions, and the order of subexpressions. Clearly, a correct definition of these properties requires a solid definition of index expressions.

Finally, a Boolean augmentation of index expressions is presented in Wondergem et al (1998d). The language of index expressions is extended with logical operators for conjunction, disjunction, and negation. The logical operators can occur in the head of index expressions as well as nested in subexpressions. This leads to a compact representation of logical combinations of index expressions. The augmented language of *Boolean index expressions* is exploited for the construction and representation of user profiles in the Profile system for information filtering (see e.g. Wondergem et al, 1997). In order for Boolean index expressions to be soundly defined, index expressions themselves must be precisely defined.

To conclude, many different applications of index expressions in IR exist. These applications presuppose a correct definition of index expressions, several notions of subexpressions, and properties of both. Although the intution of these issues may be clear, sound formal ground has not been developed yet. The main part of the remainder of this article is devoted to lifting this lack. In addition, our investigation supports the design of algorithms for computing enlargements and refinements. In particular, our defoliation operator can be directly implemented in order to obtain an algorithm for (direct) enlargements.

3. Formalizing Index Expressions

In this section, the language of index expressions is defined. Section 3.1 introduces index expressions based on the nesting operator and provides a number of properties of index expressions. Section 3.2 elaborates the defoliation of index expressions, i.e., the removal of their leaves.

This work does not aim at opening doors for new applications. Rather, next to providing a proper answer to the presumptions on which the applications of index expressions are based, it will enhance insights in the process of defoliating index expressions and derived notions of subexpressions.

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

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

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. This conforms to the original definition of index expressions as given in Bruza (1990, 1993). We will omit brackets if they are clear from the context.

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

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

An instantiation of this is obtained for $c_1 = \text{by}, I_1 = \text{industry}, c_2 = \text{of}$, and $I_2 = \text{water}$:

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 3. Language of Non-empty Index Expressions Let T be a set of terms and C be a set of connectors, such that $T \cap C = \emptyset$, 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 $\mathscr{L}^{+}_{(T,C)}$, is defined as the smallest superset of T for which

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

Here, the term h is called the *head* or *lead term*. If T and C are clear, we write $\mathscr{L}^+$ 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 Definition 6).

Example 2. Nonempty Index Expressions. Since the language of nonempty index expressions is a superset of the set of terms, i.e., $T \subseteq \mathscr{L}_{(T,C)}$, 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 $h = \mathsf{pollution}$ as head, connectors $c_1 = \mathsf{by}, c_2 = \mathsf{of}$, and $c_3 = \mathsf{in}$, and nonempty subexpressions $I_1 = E_1, I_2 = E_2$, and $I_3 = E_3$, we obtain

$$\begin{aligned} E_4 &= h \otimes_{i=1}^{3} c_i(I_i) = hc_1(E_1)c_2(E_2)c_3(E_3) \\ &= \mathsf{pollution\ by\ (industry)\ of\ (water\ and\ (air))\ in\ (rural} \circ \mathsf{(areas))} \end{aligned}$$

Nonempty index expression may not contain the empty index expression. A definition of index expressions that does allow for this is given in Berger (1998). There, the empty index expression is an example of a universal descriptor: it may represent any element of the language. The more general definition, however, allows for intuitively unclear descriptors. For example, pollution by ϵ and ϵ in ϵ are valid (generalised) index expressions. Our approach concentrates on intuitively clear index expressions only. In accordance with the original definition, there is only one use of the empty index expression. In addition, although using the generalised definition of index expressions may also lead to a more general definition of defoliation, it will also introduce additional difficulties. For example, the use of the empty index expression within nonempty ones introduces the need for normalisation in order to identify semantically equivalent index expressions.

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

Definition 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 $\mathscr{L}_{(T,C)}$, and is defined as

$$\mathscr{L}_{(T,C)} = \mathscr{L}^{+}_{(T,C)} \cup \{\epsilon\}$$

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

The size of an index expression, denoted by $|I|$ for index expression I, 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.

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 are also no leaves. This is captured in the Boolean function `IsLeaf`. For all index expressions $t \in T$, we have $\texttt{IsLeaf}(t)$.

3.2. 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 Wondergem et al (1996) and Iannella et al (1995). 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 Bruza and van der Weide (1991).

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 may occur if the index expression has exactly one subexpression. In the remainder of this article, we implicitly include this case when referring to leaves.

Pointing Sequences. The designated leaf is identified by a so called *pointing sequence*. 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. The pointing sequence [0] denotes the head of an index expression, which in the case of E_4 is pollution. Since nonempty sequences of natural numbers can identify leaves, we call them pointing sequences.

Definition 5. Nonempty Sequences of Natural Numbers Let N denote the natural numbers, i.e., $0, 1, 2, \ldots$ Furthermore, let $\mathbf{N}$ denote the set of nonempty sequences over the natural numbers. We denote a sequence consisting of a single element $x \in \mathbf{N}$ 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 3. Pointing Sequence. Figure 2 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.

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.

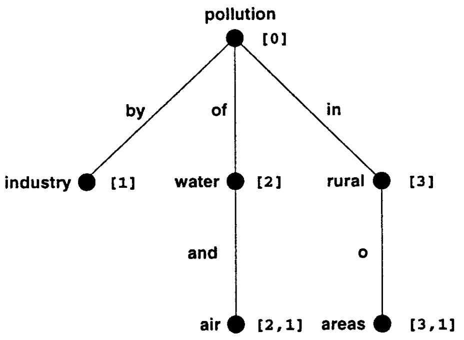

Fig. 2. Example index expressions with pointing sequences for all terms.

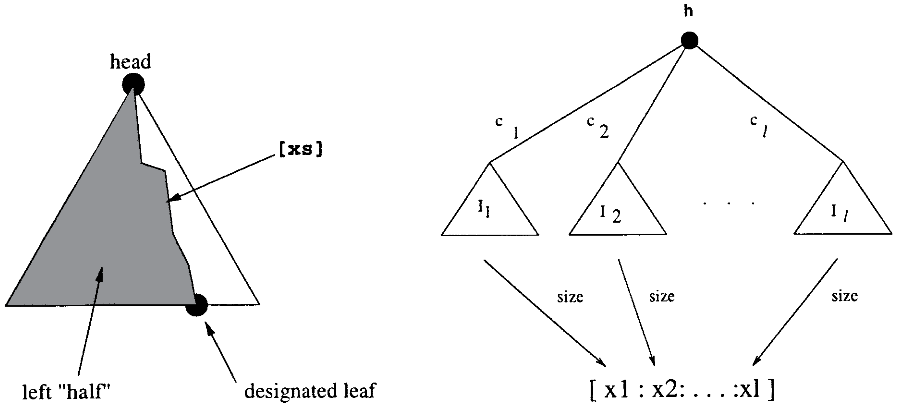

Fig. 3. (a) Pointing Sequence models Index Expression. (b) Mapping Index Expressions to Pointing Sequences.

Each leaf can be identified as the righmost 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 Fig. 3(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.

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 Fig. 3(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.

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 6. 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:

$$\Delta : \mathscr{L} \times \mathbf{N} \mapsto \mathscr{L}$$

The definition of the defoliation operator is given by (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-trival applications of defoliation. These applications remove the non-head leaf which comprises subexpression I_x. This results in the conditions $l \geqslant 1$, saying there is at least one subexpression, $1 \leqslant x \leqslant l$, 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:

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

Example 4. Non-recursive Cases of Defoliation. This example illustrates case (1), (2), and (3), respectively.

Defoliating a term results in the empty index expression:

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

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

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

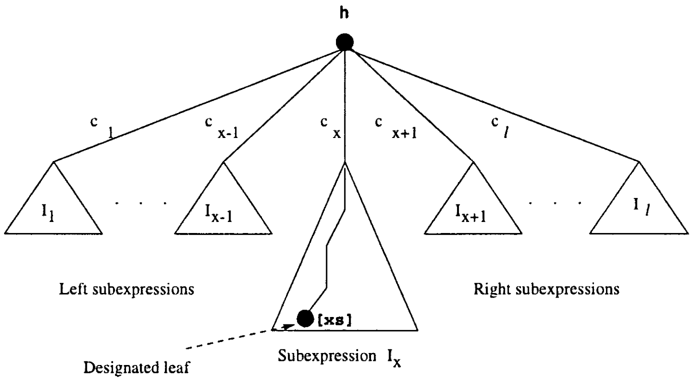

Fig. 4. Schematic description of recursive defoliation.

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

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

Consider index expression, E_4 from example 2:

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

The recursive case for defoliation is first illustrated in example 5. Then, the definition is followed by an example providing a number of practical illustrations of the recursive case of defoliation.

Example 5. Recursive Defoliation (Schematic). Figure 4 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 4 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.

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: $l \geqslant 1$. 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: $1 \leqslant x \leqslant l$. 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 subexpression right of $I_x(\otimes_{i=x+1}^{l} c_i(I_i))$ are copied:

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

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 4). The `not IsLeaf`(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 6. 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 Fig. 2).

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 4 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:

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

Example 7. Undefined Cases of Defoliation. Undefined defoliation 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.

$$\Delta(hc_1(I_1)c_2(I_2), [3, xs])$$

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

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, termination of the defoliation process is examined.

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

Lemma 7. One Term Less. *For every nonempty index expression $I \in \mathscr{L}^+$ and every pointing sequence x such that $\Delta(I, \mathrm{x})$ is defined, we have $|\Delta(I, \mathrm{x})| = |I| - 1$.*

Proof of lemma. 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, we consider the termination of a single defoliation, i.e. the removal of a single leaf. 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).

The second view on termination is as a repeated process eventually resulting in the empty index expression. 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.

4. 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.

4.1. Direct Subexpressions

The defoliation operator is now used to construct the direct subexpression relation $\prec_{\mathrm{d}} \subseteq \mathscr{L}$ for `index` expressions. Direct subexpressions of index expressions are obtained by removing exactly one leaf.

Definition 8. Direct Subexpression Relation

$$I \prec_{\mathrm{d}} J =^{\mathrm{def}} \exists \mathrm{x} \in \mathbf{N}\colon \Delta(J, \mathrm{x}) = I$$

The following lemma holds for the direct subexpression relation:

Lemma 9. $\prec_{\mathrm{d}}$ is asymmetric.

Proof of lemma. Suppose $\prec_d$ is not asymmetric, that is, $\mathtt{not}(I \prec_{\mathrm{d}} J \Rightarrow J \not\prec_{\mathrm{d}} I)$. Let I and J be such that $I \prec_d J$ and $J \prec_d I$. Then, applying Lemma 7 results in $|I| = |J| - 1$ and $|J| = |I| - 1$, which is a contradiction. □

A consequence of this lemma is that the direct subexpression relation is irreflexive.

The fact that direct subexpressions differ exactly one leaf, allows fine-grained navigation in Query by Navigation. So called one-step (direct) refinements, as proposed in Wondergem (1996), 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, i.e. direct refinements and enlargements, since the edges in lithoids coincide with the direct subexpression relation (see Bruza and van der Weide, 1992). This property is exploited in so called hyperindex browsers (for an example, see Iannella et al, 1995) 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 Wondergem (1996), where search paths impose a ranked clustering of the documents underlying the lithoid. Another example, as described in Bruza and van Linder (1997) and Wondergem et al (1996), constructs Preferential Models out of search paths. Preferential Models, which are capable of capturing (non-monotonic) user preferences, are augmented with domain knowledge in Wondergem et al (1998a).

In addition, the direct subexpression relation serves as the basis for defining the edges of more complicated navigational structures such as association indices

(Wondergem et al, 1998b,c). 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.

4.2. Strict Subexpressions

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

Definition 10. Strict Subexpression Relation

$$\prec =^{\mathtt{def}} \prec_{\mathrm{d}}^{*}$$

The following lemma is now clear:

Lemma 11. $\prec$ is asymmetric, and transitive.

Proof of lemma. Asymmetry ($I \prec J \Rightarrow J \nprec I$) follows directly from Lemma 9 and Definition 10: only transitive instances are added to $\prec_d$.

Transitivity ($I \prec J$ and $J \prec K \Rightarrow I \prec K$) follows from Definition 10: $\prec$ is the transitive closure of $\prec_d$. □

As a consequence, the strict subexpression relation is also irreflexive. Transitivity and irreflexivity of $\prec$ imply that the strict subexpression relation is a quasi order on $\mathscr{L}$.

The strict subexpression relation can be exploited for personalising search paths in QBN (Berger, 1998). This means that, for instance, short cuts can be provided to minimize navigation time. The reason for this is that it not only includes one-step refinements and enlargements. In particular, the source and destination of search paths that consist of series of 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.

4.3. General Subexpressions

The third relation on index expressions defined in this article is the (general) subexpression relation $\preceq \subseteq \mathscr{L}^2$ 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 12. Subexpression Relation

$$\preceq =^{\mathtt{def}} \prec \cup \{(I,I) \mid I \in \mathscr{L}\}$$

The following lemma holds for general subexpressions:

Lemma 13. $\preceq$ is reflexive, antisymmetric, and transitive.

Proof of lemma. Reflexivity follows from Definition 12 and Lemma 11.
To prove antisymmetry ($I \preceq J$ and $J \preceq I \Rightarrow I = J$), assume $I \preceq J$ and $J \preceq I$.

Assume $I \neq J$ for the sake of contradiction. Now, $I \leq J$ and $I \neq J$ imply $I \prec J$. This means $J \not\prec I$ (by Lemma 11) which, again with $I \neq J$, forbids $J \leq I$. Thus, $I = J$.

Transitivity follows from Definition 12 and Lemma 11. $\square$

In other words, $\leq$ is a partial order for the language of index expressions. This means that $\langle \mathscr{L}, \leq \rangle$ is a poset. In Wondergem (1996), $\langle \mathscr{L}, \leq \rangle$ 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 (Bates, 1989).

In addition, the general subexpression relation can be used to define the set of relevant documents with respect to an index expression query (Wondergem, 1996). 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.

5. Other Representations for Index Expressions

In section 3.1, 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 $\mathscr{L}_{(T,C)}$ from Definition 4.

5.1. 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 14 provides a grammar representation for index expressions. This grammar can also be used to derive the semantics of index expressions.

Definition 14. 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:

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

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

Proof of theorem.

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 non-terminal. This parse tree starts with Expr $\rightarrow$ NExpr $\rightarrow$ Term $\{\mathsf{Connector\ NExpr}\}^{l}$ for some $l \geqslant 0$. By the induction hypothesis, all $\mathsf{NExpr}_i (1 \leqslant i \leqslant l)$ 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.

   This case can be proven with induction to the number of occurrences of the nesting operator $\otimes$ in the structural representation of the index expression.

5.2. Broaden-based Representation

The broaden-based representation is defined in terms of the binary constructor operator broaden: $\mathscr{L}^{+} \times C \times \mathscr{L}^{+} \mapsto \mathscr{L}^{+}$. This operator broadens an index expression I with a connector c and an index expression J, and results in a larger index expression $\mathtt{broaden}(I, c, J)$. The broaden operator, see Fig. 5, is used in the parsing mechanism described in Bruza (1993).

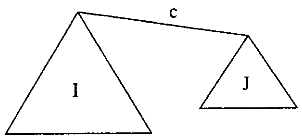

Fig. 5. Broadening an index expression; $\texttt{broaden}(I, c, J)$.

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. Furthermore, the actual structure of index expressions, i.e., lead term plus a number of subexpressions, is less clearly depicted than in the structural representation. In the structural representation, it is directly clear which subexpressions occur at the same depth. This property is exploited in, for instance, matching if all subexpressions of a certain depth are to be processed at the same time or in the same manner (Wondergem et al, 2000).

Definition 16. Broaden-based Non-Empty Index Expressions The language of non-empty index expressions, denoted by $\mathscr{L}^+$, 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 17. Broaden-based Language of Index Expressions

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

The structural and the broaden-based representation can be transformed into oneanother as sketched by the scheme below.

$$h \bigotimes_{i=1}^{l} c_i(I_i) \Leftrightarrow \texttt{broaden}(\ldots \texttt{broaden}(\texttt{broaden}(h, c_1, J_1), c_2, J_2) \ldots, c_l, J_l)$$

This scheme represents the same principle of transforming a tree into a binary tree, or vice versa. It will be exploited in the proof of the following theorem.

Theorem 18. Broaden-based $\equiv$ Structural Representation

Proof of theorem.

1. Every index expression generated by the broaden-based representation can also be described by the structural representation.
   This case can be proven similar to the first case of Theorem 15. That is, by induction to the number of occurrences of the broaden operator in index expressions.
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 17.

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 19. Broaden-based $\equiv$ Abstract Syntax. Direct consequence of Theorems 15 and 18.

6. 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 architectures 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 Berger (1998). 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

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

Berger FC (1998) Navigational Query Construction in a Hypertext Environment. PhD thesis, Department of Computer Science, University of Nijmegen, September

Berger FC, van Bommel P (1997) Augmenting a characterization network with semantical information. Information Processing & Management, 33(4):453–479

Berger FC, ter Hofstede AHM, van der Weide Th.P (1996) Supporting query by navigation. In R. Leon, ed., Information retrieval: New systems and current research, Proceedings of the 16th Research Colloquium of the British Computer Society Information Retrieval Specialists Group, pp 26–46, Drymen, Scotland, Taylor Graham

Bosman R, Bouwman, R, Bruza, PD (1991) The Effectiveness of Navigable Information Disclosure

Systems. In G. A. M. Kempen, ed., Proceedings of the Informatiewetenschap 1991 conference, Nijmegen, The Netherlands

Bruza PD (1990) 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, pp. 109–122, Cambridge, UK, Cambridge University Press.

Bruza PD (1993) Stratified Information Disclosure: A Synthesis between Information Retrieval and Hypermedia. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands

Bruza PD, van der Gaag, LC (1994) Index Expression Belief Networks for Information Disclosure. International Journal of Expert Systems, 7(2):107–138

Bruza PD, Ijdens, JJ (1994) Efficient probabilistic inference through index expression belief networks. In Proceedings of the Seventh Australian Joint Conference on Artificial Intelligence (AI94), pp 592–599. World Scientific

Bruza, PD, van Linder, B (1997) Preferential models of refinment paths. In Proceedings of the IJCAI-97 Workshop on AI and Digital Libraries

Bruza PD, van der Weide Th.P. Two level hypermedia – an improved architecture for hypertext. In A.M. Tjoa and R. Wagner, eds, Proceedings of the Data Base and Expert System Applications Conference (DEXA 90), pp 76–83, Vienna, Austria. Springer-Verlag

Bruza PD, van der Weide Th.P. (1991) The modelling and retrieval of documents using index expressions. ACM SIGIR FORUM (Refereed Section), 25(2)

Bruza PD, van der Weide Th.P. (1992) Stratified hypermedia structures for information disclosure. The Computer Journal, 35(3): 208–220

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

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

van Rijsbergen CJ (1979) Information Retrieval. Butterworths, London, UK, 2nd edition

Winograd W (1983) Language as a Cognitive Process. Addison-Wesley Pub. Co., Reading, MA, USA

Wondergem BCM (1996) Preferential structures for information retrieval. Master's Thesis INF-SCR-96-21, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, August

Wondergem BCM, van Bommel P, Huibers TWC, van der Weide Th.P (1998a) Domain knowledge in preferential models. In Janis Barzdins, ed., Proceedings of the Third Baltic Workshop DB& IS98, volume 1, pp 126–138, Riga, Latvia, April

Wondergem BCM, van Bommel P, van der Weide Th.P (1998b) Association index architecture for information brokers. Technical Report CSI-R9820, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, July

Wondergem BCM, van Bommel P, van der Weide Th.P (1998c) Construction and applications of the association index architecture. In *Proceedings of CIW'98, the Conferentie Informatiewetenschappen 1998*, Antwerp, Belgium, December

Wondergem BCM, van Bommel P, van der Weide Th.P (1998d) Boolean index expressions for information retrieval. Technical Report: CSI-R9827, University of Nijmegen, Nijmegen, The Netherlands, December

Wondergem BCM, van Bommel P, van der Weide Th.P (2000) Matching index expressions for information retrieval. Information Retrieval Journal. To appear

Wondergem BCM, van der Hoek W, Huibers TWC, Witteveen C (1996) Preferential semantics for query by navigation. In K van der Meer, ed., *Informatiewetenschap 1996*, pp 153–168, Delft, The Netherlands, December

Wondergem BCM, Simons J, Arampatzis AT, Mackowiak J, Tarenskeen D., Huibers TWC (1997) Profile information filtering project – overall project plan. Technical Note CSI-N9707, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, September

Author Biographies

Bernd Wondergem received his masters degree in Computer Science in 1996 from Utrecht University, the Netherlands. He is currently a PhD student at the University of Nijmegen, the Netherlands.

His main research interests include information retrieval and filtering and knowledge-based systems. Major issues within these fields include expressive but tractable descriptor languages and logical models for retrieval. His PhD research is done within the Profile project (Wondergem et al, 1997), which aims at developing a proactive information filter for the WWW.

P. van Bommel received his masters degree in Computer Science in 1990, and the degree of PhD in Mathematics and Computer Science, from the University of Nijmegen, the Netherlands in 1995. He is currently assistant professor at the University of Nijmegen, the Netherlands.

His main research interests include information modelling and information retrieval. In information modelling, he particularly deals with equivalence and mapping of data models, the transformation of (conceptual) data models into efficient (internal) representations, and the corresponding transformation of database populations, operations and constraints. In information retrieval his main research interests include WWW-based retrieval/filtering, and document characterisation languages.

Ir. Th. P. van der Weide received his masters degree from the Technical University Eindhoven, the Netherlands in 1975, and the degree of PhD in Mathematics and Physics from the University of Leiden, the Netherlands in 1980.

He is currently associate professor at the University of Nijmegen, the Netherlands. His main research interests include information systems, information retrieval, hypertext and knowledge-based systems.

Correspondence and offprint requests to: Bernd Wondergem, Computing Science Institute, University of Nijmegen Toernooiveld 1, NL-6525 ED, Nijmegen, The Netherlands.