Compact and Tractable Descriptors for Information Discovery

een wetenschappelijke proeve op het gebied
van de
Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Katholieke Universiteit Nijmegen,
volgens besluit van het College van Decanen in het
openbaar te verdedigen op maandag 2 oktober 2000
des namiddags om 1:30 uur precies

door

Bernd Christian Martijn Wondergem

geboren op 5 februari 1973 te Ede.
Promotor: Prof. C.H.A. Koster

Co-pomotores: Dr. ir. Th.P. van der Weide
               Dr. P. van Bommel

Manuscriptcommissie: Prof. dr. C.J. van Rijsbergen (University of Glasgow)
                      Prof. dr. N. Fuhr (Universität Dortmund)
                      Dr. P.D. Bruza (University of Queensland)

ISBN 90-6464-831-X
NUGI 851
2000
## Contents

1 Introduction ......................................................... 11
   1.1 Properties of Descriptor Languages ....................... 11
   1.2 PROFILE Research Context ................................. 13
   1.3 Intention of this Thesis ...................................... 14
       1.3.1 Problem Statement .................................... 14
       1.3.2 Scope of the Thesis .................................... 16
   1.4 Structure of this Thesis ...................................... 17
       1.4.1 Overview of Thesis .................................... 17
       1.4.2 Paths of Reading ...................................... 17

2 Information Discovery ............................................. 19
   2.1 Introduction .................................................... 19
   2.2 Information Retrieval .......................................... 19
       2.2.1 Information Retrieval Paradigm ...................... 20
       2.2.2 History of Information Retrieval ...................... 21
       2.2.3 Information Retrieval Models ......................... 23
   2.3 Information Filtering .......................................... 30
       2.3.1 Persistency of Interests ............................... 31
       2.3.2 Stream of Documents ................................. 31
       2.3.3 Interaction .............................................. 31
       2.3.4 Social Aspects .......................................... 32
   2.4 The Information Discovery Paradigm ...................... 32
       2.4.1 Benefits of the Synthesis ............................. 33
       2.4.2 Agent-Based Information Discovery .................. 34
3 Profile - A Multi-Disciplinary Approach to ID

3.1 Introduction ................................................................. 47

3.2 Profile Requirements .................................................... 48
   3.2.1 Constraints on Profile ........................................ 48
   3.2.2 Properties of Agents ........................................... 49
   3.2.3 Overall System Design ....................................... 50

3.3 Research in Profile ....................................................... 51
   3.3.1 User Modeling .................................................... 51
   3.3.2 Language Processing ......................................... 53
   3.3.3 Retrieval .......................................................... 55
   3.3.4 User Interface .................................................... 57

3.4 Current Prototype ........................................................ 58
   3.4.1 Architecture ....................................................... 58
   3.4.2 Agents in the Implementation .............................. 58
   3.4.3 Working of the Implementation ........................... 59

3.5 Outlook ........................................................................ 60
4 Index Expressions
  4.1 Introduction .................................................. 63
  4.2 Fundamentals of Index Expressions ......................... 63
  4.3 Nesting and Defoliation ...................................... 68
    4.3.1 Structural Representation of Index Expressions .......... 68
    4.3.2 Defoliation of Index Expressions ...................... 71
  4.4 Representational Formalisms ................................ 78
    4.4.1 Structural Representation ................................ 78
    4.4.2 Grammar Representation ................................. 79
    4.4.3 Inductive Broaden-Based Representation ............... 81
  4.5 Subexpressions .............................................. 84
    4.5.1 Direct Subexpressions ................................. 84
    4.5.2 Strict Subexpressions ................................. 85
    4.5.3 General Subexpressions ............................... 86
    4.5.4 Embedding ............................................. 86
  4.6 Matching ................................................... 87
    4.6.1 Introduction ........................................... 87
    4.6.2 Related Work ........................................... 88
    4.6.3 Criteria for Matching .................................. 89
    4.6.4 Similarity Measures for Index Expressions ............ 91
    4.6.5 Experimental Evaluation ............................... 98
    4.6.6 Outlook on Matching ................................... 105
  4.7 Additional Topics concerning IEs .......................... 105
    4.7.1 Relation with Noun Phrases ............................ 105
    4.7.2 Obtaining Index Expressions from Text ................ 106
    4.7.3 Counting Index Expressions ........................... 107
  4.8 Outlook .................................................... 109

5 Boolean Index Expressions ...................................... 111
  5.1 Introduction .................................................. 111
  5.2 Boolean Index Expressions .................................... 111
    5.2.1 Representation of BIEs .................................. 112
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2 Properties of BIEs</td>
<td>114</td>
</tr>
<tr>
<td>5.2.3 BIEs for Retrieval and Filtering</td>
<td>115</td>
</tr>
<tr>
<td>5.3 On Compactness of BIEs</td>
<td>118</td>
</tr>
<tr>
<td>5.4 Zipping Operators in BIEs</td>
<td>119</td>
</tr>
<tr>
<td>5.4.1 Priority Scheme</td>
<td>120</td>
</tr>
<tr>
<td>5.4.2 Purpose of Zipping</td>
<td>121</td>
</tr>
<tr>
<td>5.4.3 Zipping Functions</td>
<td>123</td>
</tr>
<tr>
<td>5.4.4 Result of Zipping</td>
<td>126</td>
</tr>
<tr>
<td>5.4.5 Dyadic Congruence and Equivalence</td>
<td>129</td>
</tr>
<tr>
<td>5.5 Expansion by Zipping Operators</td>
<td>137</td>
</tr>
<tr>
<td>5.5.1 Dyadic Count as Measure for Expansion</td>
<td>137</td>
</tr>
<tr>
<td>5.5.2 Other Measures for Expansion</td>
<td>143</td>
</tr>
<tr>
<td>5.6 Minimal Expansion</td>
<td>144</td>
</tr>
<tr>
<td>5.6.1 Explicit Characterisation of Minimal BIEs</td>
<td>144</td>
</tr>
<tr>
<td>5.6.2 Lower Bound</td>
<td>146</td>
</tr>
<tr>
<td>5.6.3 Example Minimal BIEs</td>
<td>148</td>
</tr>
<tr>
<td>5.6.4 Quantitative Analysis of Minimal BIEs</td>
<td>149</td>
</tr>
<tr>
<td>5.7 Maximal Expansion</td>
<td>152</td>
</tr>
<tr>
<td>5.7.1 Maximal Positive BIEs</td>
<td>153</td>
</tr>
<tr>
<td>5.7.2 Maximal BIEs</td>
<td>159</td>
</tr>
<tr>
<td>5.7.3 Quantitative Analysis of Maximal BIEs</td>
<td>164</td>
</tr>
<tr>
<td>5.7.4 Dealing with Impractical Expansion</td>
<td>166</td>
</tr>
<tr>
<td>5.8 Applying BIEs in ID</td>
<td>169</td>
</tr>
<tr>
<td>5.8.1 Obtaining BIEs from Text</td>
<td>169</td>
</tr>
<tr>
<td>5.8.2 Matching BIEs</td>
<td>169</td>
</tr>
<tr>
<td>5.9 Implementation</td>
<td>172</td>
</tr>
<tr>
<td>5.10 Outlook</td>
<td>175</td>
</tr>
<tr>
<td>6 INdex Navigator</td>
<td>177</td>
</tr>
<tr>
<td>6.1 Navigational Query Formulation</td>
<td>177</td>
</tr>
<tr>
<td>6.2 Related Approaches</td>
<td>178</td>
</tr>
<tr>
<td>6.3 Stratified Architecture for Index Expressions</td>
<td>179</td>
</tr>
</tbody>
</table>
CONTENTS

6.4 Query by Navigation ................................................................. 180
  6.4.1 Navigating the Hyperindex .................................................. 180
  6.4.2 Beaming between Layers .................................................... 182
6.5 INN System .................................................................................... 182
  6.5.1 Getting Started ................................................................. 183
  6.5.2 Navigating on the Fly ......................................................... 185
  6.5.3 Beaming between Layers .................................................... 187
6.6 Experiences with the INN System .................................................. 187
  6.6.1 When No Relevant Documents are Available .................... 187
  6.6.2 Dependency on Search Engines ........................................... 187
6.7 Association Index Architecture .................................................... 188
  6.7.1 Architecture ................................................................. 188
  6.7.2 Applications of the AIA ...................................................... 190
6.8 Outlook ....................................................................................... 191

7 Conclusions and Further Research .................................................. 193

Bibliography .................................................................................. 197

Index ............................................................................................ 212

Curriculum Vitæ ............................................................................ 217
Dankwoord

Dit proefschrift is tot stand gekomen dankzij de hulp en inzet van een groot aantal personen.

Allereerst dank ik mijn promotor Kees Koster en co-promotoren Patrick van Bommel en Theo van der Weide. Zij hebben een onmisbare inhoudelijke en procesmatige rol gespeeld bij mijn onderzoek. De wekelijkse besprekingen met Patrick zorgden voor directe feedback. In het bijzonder dank ik Patrick voor zijn inzet en de vrijheid die hij me gaf. Theo wil ik met name bedanken voor zijn visie en hulp bij het laatste jaar van mijn onderzoek.

In het eerste jaar van mijn aio-schap was ook Theo Huibers mijn begeleider. Theo wil ik graag bedanken voor zijn aanstekelijk werkende energieke aanpak. Ook ben ik hem zeer dankbaar voor het netwerk aan contacten dat ik via en middels hem heb opgebouwd.


Een leuk onderdeel van mijn werkzaamheden was het begeleiden van afstudeerders. Ik wil met name Pim van Mun and Mark van Uden bedanken voor hun inzet en de prettige samenwerking. Concrete resultaten uit hun werk zijn opgenomen in dit proefschrift.

De leescommissie heeft in de eindfase een belangrijke rol vervuld door mijn manuscript van commentaar en suggesties te voorzien. Naast de reeds genoemde personen wil ik graag Franc Grootjen, Ger Paulussen en Janos Sarbo bedanken voor hun constructieve en gedetailleerde opmerkingen.

De leden van het PROFILE project wil ik bedanken voor de prettige samenwerking. Hierbij denk ik aan Avi Arampatzis, Joep Simons, Janek Mackowiak, Debbie Tarenskeen en de leden van de stuurgroep.

Tenslotte wil ik ook graag mijn familie en vrienden bedanken. In het bijzonder wil ik Aukje bedanken voor haar aandacht en steun. Naast het vele plezier dat jullie me boden, waren ook de afleiding en rust erg belangrijk voor me.

Bernd
Chapter 1

Introduction

The advent of computers and especially the Internet has facilitated an enormous increase in the amount of electronically available information. The quantity and diversity of currently available information is like never before. However, the difficulty of finding relevant information has grown accordingly.

The goal of Information Retrieval (IR) ([Rij79]) is to provide users effective and simple access to large quantities of unstructured information. This information originally was only of textual nature, but today often is multi-medial. Information Filtering (IF) can be seen as a dual approach with the same goal as IR ([BC92]). In both approaches, descriptor languages play an important role.

Descriptor languages form a key ingredient for the three main processes in IR and IF. First, the user formulates his information need(s) in terms of (several) descriptors. Second, the contents of documents is described by descriptors. Third, similarity between descriptors forms the basis for relevance estimates. Relevance estimates state which documents are expected to be relevant to which information need. Retrieval and filtering systems render documents based on relevance estimates.

The remainder of this chapter is organised as follows. Section 1.1 discusses three important properties of descriptor languages. Then, Section 1.2 sketches the context in which the research reported in this thesis was conducted. The intention of this thesis is described in Section 1.3. Finally, Section 1.4 provides the structure of the remainder of this thesis.

1.1 Properties of Descriptor Languages

Formulating an information need by summing up its features by keywords (the bag-of-words model) has not proved a successful approach, as has been generally recognised in literature (e.g. [SJT84, Str95, Sme97]) and practice ([VH96]). Although a major advantage of the bag-of-words model is computational simplicity, or, tractability, it does not support a reasonable approximation of any underlying more or less sophisticated cognitive model.
In other words, the *expressiveness* of the bag-of-words model does not support an adequate communication mechanism for information needs. Several approaches have been tried to go beyond the bag-of-words model by explicit inclusion of relations between features.

As a simple attempt forward, the Boolean keyword model includes logical relations between features. The AND operator is the, rather crude, analogue for phrase construction while the OR operator allows for simple inclusion of related terms such as synonyms and morphological variants. Furthermore, the model also allows for formulation by exclusion (i.e. by using negations). The Boolean keyword model, however, still leaves room for many ambiguities by insufficient (linguistic) expressiveness and suffers from lack of comprehensibility ([Coo88, SFW83]).

As more sophisticated, linguistically motivated descriptors, noun phrases have been proposed. In artificial intelligence, noun phrases are considered as references to or descriptions of complicated concepts ([Win83]). Noun phrases are generally accepted to improve precision by lowering the number of irrelevant rendered documents. However, the price for these improvements, besides being computationally less tractable ([Str95]), often is a simultaneous decrease of recall (e.g. [SJT84]), meaning that more relevant documents are missed. This suggests that noun phrases are too expressive for retrieval and filtering ([Sme97, AWKB00b]).

As an intermediate form, index expressions (see e.g. [BW92, Bru93, WBW00]) have been introduced as a simple yet powerful approximation of noun phrases. Index expressions describe a concept by a general term, further constrained by modifying subexpressions. A modifying subexpression is connected to the general term by a connector (e.g. preposition or present participle), which expresses the relation between the general term and the modifying concept. Index expressions have been applied in IR for supporting query formulation ([Bru90, BB97]), document indexing ([BW91]), matching ([WBWb]), and the construction of layered hypertext architectures ([BW90]). However, index expressions lack a logical mechanism for concept construction, suggesting too limited (logical) expressiveness. Furthermore, index expressions do not support a *compact* representation of information needs. Compactness offers the opportunity to convey much information in a succinct way.

Descriptor languages for filtering and retrieval should find a workable balance between the following properties:

**Expressive.** Expressiveness of a descriptor language is viewed as its ability to discriminate between different sets of documents. In general, this is directly dependent on the variety of descriptors in the language.

Expressiveness is particularly relevant for IF where user profiles are assumed to form a correct representation of the information need of those users ([BC92]). Expressiveness is generally acknowledged as beneficial for precision but also to cause a drop in recall ([SJT84]). Several normalisation techniques (see e.g. [ATKW98, Str95]) can be applied to attack the negative influence of expressiveness on recall.

The use of complex descriptors in IR has led to mixed empirical results. Significant improvements over the bag-of-words model were reported in, for instance, [AWKB00a],
However, others reported no improvements in effectiveness (see e.g. [Lew92]). This thesis focuses on compactness enabled by the structure of descriptors.

**Tractable.** Tractability, or simplicity of processing, is viewed as a system criterion. Tractability is inversely proportional to the difficulty to solve a problem. The meta-term intractable is often used to express that all known solutions for a problem require exponential time ([GJ79]). We consider tractability of a descriptor language as inversely proportional to the amount of time, space (computer memory), and auxiliary knowledge required to process them in IR and IF applications.

**Compact.** Compactness is the ability to convey much information in a succinct way. From a user-oriented point of view, compactness is beneficial in formulating the information need. If an information need involves several related concepts, subexpression sharing saves space (for representing the need) and time (for users to read the representation). In this sense, compactness has not (yet) received a lot of research attention in the fields of IR and IF. We argue, however, that the growing interest in persistent information needs leads to the (stepwise) formulation of more complex information needs. This, then, increases the usability of compact descriptors.

Before stating the main research theme of this thesis in terms of the properties of descriptor languages mentioned above, we describe the context in which the research was conducted.

### 1.2 Profile Research Context

The research reported in this thesis was done in the context of the Profile ([HSB+96, WSA+97, SAW+00]) project. Profile is a multi-disciplinary research project aiming at the development of a proactive filtering system as effective intermediary between users and sources of information. In the Profile system object-triggered (filtering) and user-triggered (retrieval) information interchange are to be integrated.

The Profile project consists of four components. These are labeled user modelling, language processing, user-computer interaction, and retrieval. The picture in Figure 1.1 roughly illustrates the cyclic interaction between the four Profile components. First, users formulate their information need by interacting with the Profile system. Based on user behaviour, the user modelling component derives a user model. The parsing model then translates the user model to descriptors. In addition, it indexes documents by sets of descriptors, which are used in the matching functions of the retrieval component. Finally, the selected documents are presented to the user. Upon rendering of documents, the user may provide relevance feedback by interacting with the system. This may lead to the adaptation of the user profile and start a new Profile information cycle.

This thesis describes the work done for the retrieval component of the Profile project.
1.3 Intention of this Thesis

The main theme of the research reported in this thesis can be summarised as follows:

*Design a descriptor language that forms a workable balance between expressiveness and tractability and which supports compact representations of information needs.*

Section 1.3.1 describes the research questions related with this theme in more detail. Subsequently, the scope of this thesis is provided in Section 1.3.2.

1.3.1 Problem Statement

This section describes the research questions that are investigated in this thesis.

**Synthesising Retrieval and Filtering** Retrieval and filtering are two main approaches to the quest for information. Combining both approaches in a single system allows a powerful and uniform interface to the information environment at hand. Therefore, the benefits and difficulties of synthesising IR and IF in a single paradigm are to be explored. Current information environments like the internet are highly distributed and consist of numerous connected information sources. The resulting paradigm should adequately describe distributed information environments featuring multiple users and sources of information.

**PROFILE Architecture** A multi-disciplinary research project like PROFILE poses constraints of flexibility and modularisation upon the architecture that is used for implementation. Therefore, the suitability of an agent-based architecture for the PROFILE

![Figure 1.1: PROFILE Information Cycle.](image-url)
1.3. INTENTION OF THIS THESIS

The project is to be evaluated. This should be done in the light of the project as a whole, i.e., considering the research of the four components of the PROFILE project. The evaluation should have a general setting, considering functional system requirements as well as organisational issues.

Foundation of Index Expressions Index expressions form the starting point for our research into compact descriptors. Therefore, a solid formal basis for index expressions is required. Many applications of index expressions hinge on notions of subexpressions. For example, the core of the stratified architecture for index expressions requires direct subexpressions to be available. Different notions of subexpressions may be obtained by repeated removal of leaves, or, defoliation. Therefore, defoliation and nesting of subexpressions should be well-specified in the formal representation of index expressions.

Similarity between Index Expressions Similarity measures form key functions in many retrieval and filtering tasks. In order to enhance the applicability of index expressions in information retrieval and filtering, numerical similarity measures are to be designed. These similarity measures should focus on the refinement structure of index expressions. Because of our aim to exploit the similarity functions in a dynamic hypertext system, they should not require large auxiliary knowledge bases. The workings of the similarity functions are to be evaluated in the context of subexpressions since these form the basis for navigational networks for index expressions. Such networks are part of the dynamic hypertext system.

Compact Descriptors A compact descriptor language that is also expressive and tractable is to be designed. This is to be done by augmenting index expressions with Boolean operators for disjunction, conjunction, and negation. The augmented descriptors are called Boolean index expressions (BIEs). Nested occurrences of the Boolean operators should be allowed. This enables subexpression sharing and effectuates compact representations. A clear insight in the nature of compactness is required. Therefore, the borders of compactness are to be examined qualitatively and quantitatively.

Reducing Syntactical Variety The Boolean operators in BIEs introduce syntactical variety. To deal with syntactically different but semantically equivalent BIEs, a normalisation procedure for BIEs is to be devised. The normal form for BIEs should express the same information as a logical combination of elementary BIEs. The reason for this is to bypass the need to characterise documents with full-fledged BIEs. Rather, indexing documents with index expressions then suffices for applying BIEs for formulating information needs.

Navigational Query Construction in Dynamic Environments Navigational query formulation is a viable approach in the stratified architecture for index expressions. However, the size and dynamics of recent information environments like the World Wide Web (WWW) prohibit such an architecture to be completely created and kept up to
date. It is to be investigated how navigational query formulation can be made appli­
cable for very large and dynamic information environments. The resulting dynamic
hypertext system should be implemented and made available via the WWW.

1.3.2 Scope of the Thesis

This section sketches some boundaries on the contents of this thesis. The multi-disciplinary
nature of the PROFILE project, covering at least four research areas, may trigger the ex­
pectation of interested readers to find particular topics discussed in this thesis. Out of
necessity, we have to limit ourselves to a number of topics. Below follows a description of
some topics that will not be covered in this thesis.

Agent Technology. Agent technology, within the PROFILE project, is considered as a
means to model and implement our distributed information discovery system, not as
a research goal by itself. We examine several aspects that deal with the interface
between information discovery and agent technology. We do not, however, aim at
research that is fundamental for agents.

Linguistic Theory. We do not elaborately treat linguistic theory in this thesis. Within
the PROFILE project, the language processing component is mainly responsible for
issues concerning natural language. For example, parsing descriptors and linguisti­
cally motivated normalisation are topics of the language processing component. We
do, however, indicate relevant linguistic results.

Formal Semantics. A formal semantics of descriptor languages is not developed in this
thesis. Rather, indirect semantics are used, showing the meaning of a descriptor in
terms of the documents it renders. This approach is collection-dependent and depends
on the way in which similarities between query and document characterisation are
computed. In addition, part of the semantics of BIEs is specified by the normalisation
procedure.

Recall/Precision Graphs. No recall-precision graphs are presented in this thesis. Rather,
the virtues of our work are described by qualitative properties. In addition, we pro­
vide implementations of our work and evaluate it experimentally. The experiments
center around key issues concerning the descriptor languages.

Cognitive Evaluation. A study into cognitive aspects of (benefits of) compact descrip­
tors and BIEs in particular is outside the scope of this thesis. Such research could
focus on additional properties of descriptor languages such as comprehensibility. This
remains an issue for further research.
1.4 Structure of this Thesis

Section 1.4.1 provides an overview of this thesis. Section 1.4.2 suggests several ways of reading this thesis.

1.4.1 Overview of Thesis

The remainder of this thesis has the following structure. Chapter 2 gives a short introduction to information retrieval and filtering. Furthermore, it describes their synthesis in the information discovery (ID) paradigm. In addition, it sketches the role of agents in ID. Special attention is paid to information brokers: agents that form intermediaries between user and source agents. Finally, the influence of the duality between filtering and retrieval on the design of information brokers is described. Each chapter is ended by a section that provides directions for further research for that specific topic.

Chapter 3 elaborates on the suitability of an agent-based architecture for the Profile project. It also provides an overview of the research done by the four project components. Furthermore, it describes the integration of this research which has led to the design of a prototype system. Issues concerning the implementation of the prototype are elaborated.

Chapter 4 focuses on the use of index expressions in ID. First, a state of the art overview is given. Then, a formal basis for index expressions is developed. This consists of an investigation of nesting subexpressions and defoliation, a comparison of different representational formalisms, and the design and evaluation of several matching functions for index expressions.

Chapter 5 elaborates on the language of Boolean index expressions. The implications of compactness for use in ID are illustrated. Furthermore, normalisation of BIEs is covered by a process called zipping. Zipping translates BIEs into equivalent expressions in which disjunctions and conjunctions are unnested. It is shown that this results in BIEs in disjunctive normal form. Zipped BIEs may require a larger number of constituents than the original. This is called expansion and forms the basis for our investigation into compactness. Minimally and maximally compact BIEs are explicitly researched. Furthermore, it is shown how BIEs can be matched with (sets of) index expressions. This opens the door for applying BIEs in ID. Finally, our implementation of BIEs is described.

Chapter 6 describes a dynamic hypertext system for the WWW, the INdex Navigator (INN). The INN applies navigational query construction, as supported by the stratified architecture for index expressions, on the WWW. The system is implemented and available through the Internet.

1.4.2 Paths of Reading

There are several ways of reading this thesis. Three of them are indicated in the precedence graph of Figure 1.2. To get a quick impression, one may read Chapters 1, 2, 3, and
7. Chapter 2 sketches the context of the research reported in this thesis. It is based on [WHHW96, WBHW97a, WBHW97b, WBHW98a, WBHW98b, WBHW98c, WBW98d, WBW99b]. Chapter 3 describes the project in the light of which the research was done. It is based on [WSA+97, SAW+00]. Chapter 7 evaluates the research goals set for this thesis and provides concluding remarks.

A second way of reading this thesis may focus on the use of index expressions in information discovery. In addition to the chapters of the quick impression, it includes Chapters 4 and 6. Chapter 4, describing the use of index expressions in information discovery, is based on [WBWb, WBW00]. Chapter 6, providing the INdex Navigator, is based on [WBW98a, WBW98b, WUBW00].

Reading the complete thesis is the third way. It includes Chapter 5, which describes the compact and tractable language of Boolean index expressions. Chapter 5 is based on [WBWa, WBW98c, WBW99a].
Chapter 2

Information Discovery

Due to the proliferation of computing and networking, the desires of almost everyone to be interconnected, and the needs to make data accessible at any time and any place, modern information environments have become large, open, and heterogeneous.

M.N. Huhns and M.P. Singh, 1997, [HS97]

2.1 Introduction

In the quest for relevant information, two approaches have emerged: Information Retrieval and Information Filtering. In dynamic information environments like the World Wide Web, both approaches are valuable. Therefore, a new paradigm is introduced consisting of the synthesis of both previous approaches. This is what we coin Information Discovery (ID). An essential feature of ID is mediated information delivery, supported by information brokers. Careful broker design is required to cater for privacy of user interests, for instance. By elaborating on issues concerning ID, this chapter describes the context of our research.

The structure of this section is as follows. In Section 2.2, Information Retrieval is elaborated upon. Section 2.3 describes the dual approach of Information Filtering. Section 2.4 introduces the paradigm of Information Discovery. Section 2.5 touches upon the design of information brokers, which act as intermediaries in Information Discovery. Finally, section 2.6 provides suggestions for further research.

2.2 Information Retrieval

The goal of Information Retrieval (IR) is to provide users with effective and simple access to large quantities of unstructured information. This information originally was only of textual nature, but today often is multi-medial.
2.2.1 Information Retrieval Paradigm

IR considers a user having an information need. This information need is to be satisfied by information, referred to as documents. The documents reside in some stable document collection. To obtain this goal, IR links users to (producers of) information, as sketched in Figure 2.1. IR acquires itself of this task by the following procedure. As sketched on the left hand side of Figure 2.1, the user formulates his information need into a search request, called a query. From this query, a representation is derived which is used for matching with document contents. As sketched on the right hand side of the figure, documents are produced or delivered by some producer of documents. Since documents as a whole are rather impractical for IR, the contents of documents are described more explicitly but less precisely. This is done by a process called indexing or characterising. Characterisations of documents are organised into representations. Then, the representation of the query is matched against those of the document characterisations. Documents having a high similarity with the query are considered as relevant and are presented to the user.

![Figure 2.1: Information Retrieval Paradigm.](image)

In reaction to the documents presented, the user may express his preferences. The user may indicate implicitly or explicitly which of the documents presented are relevant to his need and which are not. This is called relevance feedback and is used by the system to produce better search results. Often the query is adapted (automatically) on the basis of relevance feedback. The adapted query in turn results in a set of documents, in reaction to which the user may again provide relevance feedback. This iterative process is repeated
until the user is satisfied with the rendered documents.

Perceiving the rendered documents may alter the user’s information need. That is, by learning, the information need may evolve or develop. In general, interaction and iterative formulation are exploited to deal with developing information needs ([BOB82]).

2.2.2 History of Information Retrieval

Automatic disclosure of information commences after the Second World War. The publication of Vannevar Bush’s visionary article “As we may think” ([Bus45]) was a strong incentive for theoretical and practical work in the field. Dr. Vannevar Bush (1890-1974) was scientific advisor of president Roosevelt. Led by scientific developments of the time and in search of new goals for post-war science, Bush looks into the future and foresees Memex, a personal and automated extension on human memory. Memex supports associative selection, adaptation of data to personal desires and combination of several texts. Currently, decades later, adaptive hypermedia systems equipped with sophisticated retrieval techniques seem to form a computerised implementation of Bush’s ideas.

Scientific developments in IR can roughly be divided into two phases ([SJW97]), as illustrated in Figure 2.2.

![Figure 2.2: Historic overview of research in IR.](image-url)
Phase 1: Development of Ideas and Techniques

In the first phase, ranging from “Bush” until the mid-seventies, scientific activities in IR center around developing important ideas and techniques.

Although Mooers introduces the term “Information Retrieval” in 1952 ([Moo52]), the specialty at that time is better addressed as “Data Retrieval”. The reason for this is that information about documents — such as author, title, and place-code — was searched, rather than information in documents. Document content is only taken into account on a large scale since the development of automatic indexing techniques. This was done at IBM in the late fifties. As suggested by Sparck Jones ([SJW97]), the ICSI conference in Washington (1958) marks the start of IR as we know it.

Several years earlier, measures to express the effectiveness of IR systems had been introduced. These are recall, signalling how well the system locates relevant documents, and the pertinency factor, expressing how well the system leaves out irrelevant documents. Around 1965, the term pertinency factor is replaced by precision, and is still in use today.

Several models for IR were proposed in the first phase. These will be elaborated in the next section. Most IR systems from the initial stage use Boolean operators (AND, OR, and NOT) to structure queries. In the early sixties, the Vector Space Model (VSM) is introduced, having a geometric basis rather than a logical one. Around the same time, the first statistical approaches to IR start. However, it would last until 1977 before the famous “Probability Ranking Principle” was posited ([Rob77]).

Meanwhile, large scale experiments for IR systems were set up. The Cranfield tests ([Cle67]) are seen as milestone in the evaluation of IR systems. Rocchio’s idea of relevance feedback ([Roc71]) completes the basis for IR. The remainder of the first phase was used to work out the ideas and to ascertain their merits.

Phase 2: Bringing IR into operation

In the second phase, running from the mid-seventies, IR is brought into operation. Also in extra-scientific scenes IR is viewed as an important specialty. Although the roots of IR lie in library science, the share of computer science gradually increases. Research into IR steadily proceeds, which results in the first SIGIR (Special Interest Group on Information Retrieval) conference in 1978. The SIGIR conferences have developed into the most important within the specialty.

The eighties show some new or revived approaches. One of the most important ones is the use of natural language processing (NLP). In the initial era of IR, some research into NLP was conducted but issues of capacity (slow computers, little memory) and the relative success of other approaches kept NLP research restricted at that time.

In the early nineties, research into IR assumed enormous proportions. One major incentive is the introduction of the World Wide Web. Information of multimedia nature, including sound, images, and video, becomes part of documents. Information is often offered in a
distributed manner and is of heterogeneous form and content. The need for interdisciplinary research becomes increasingly apparent. The IR paradigm as sketched in Figure 2.1, needs to be reconsidered in the light of this. New paradigms originate, including multiple users, sources, and intermediaries.

In 1992, the first TREC (Text REtrieval Conference) was held, a competition amongst IR systems. TREC has become a yearly event and has acquired substantial prestige.

2.2.3 Information Retrieval Models

In the course of time, many different models for IR have been developed. Generally, a model for IR describes the way in which query and characterisations are obtained and represented and how they are matched.

An advantage of model-based IR is that underlying assumptions are made explicit, which facilitates analysis and rationalisation. For instance, it eases the design of explanatory models for IR. In addition, IR systems can be compared by properties of their models ([Hui96]). The history of IR has shown a cross fertilisation between model-based and experimental approaches to IR.

The remainder of this section introduces the main model-based approaches to IR. The given classification is not strict since there are strong relationships between the models described and hybrid forms are also possible. Furthermore, the models can be thought of as classes since often many instantiations are still allowed, filling in the details in different ways. The first three models are often viewed as the basic models for IR. Subsequently, we touch upon advanced logical models, inference networks, natural language models, and cognitive models.

The Boolean Model

The underlying assumption of the Boolean Model (BM) is: A document is considered relevant to a query if the query can be logically derived (inferred, concluded) from the document’s characterisation. To facilitate this, query and characterisations are represented as Boolean expressions. The expressions consist of atoms (terms) and Boolean operators, allowing conjunction (AND), disjunction (OR), and negation (NOT). Document characterisations generally only contain (implicit) AND operators.

Logical expressions of Boolean form can be efficiently processed by implementing them as series of bits. This allows a fast bitwise computation. This is one of the reasons of the initial popularity of the BM for practical applications. From a formulation point of view, the BM suits users who know exactly what they want and how to specify it. Furthermore, synonymic and phrasal relations are readily included in the BM by expressing them as disjunctions and conjunctions, respectively.

However, the BM has several well observed limitations (see e.g. [Coo88]). First, information needs are often hard to express in Boolean form since their semantics may be complicated
and hard to grasp. However, trained users, having handiness with Boolean constructs, may view this expressiveness as an advantage. For instance, a conjunction of a small number of properly selected terms may already deliver a well-filtered document set. ANDed terms are highly discriminating and effectively filter out noise. Second, it is impossible in the original BM to deliver the resulting documents in ranked order. The reason for this is that the BM supports a binary relevance computation: documents are either considered relevant or not. This is why the BM is called an exact match model. Third, the BM offers very little control over the size of the result set. Without detailed knowledge about the spreading of terms in the document collection, a priori estimates are hard to make. Finally, the relative importance of the components of queries is hard to express in the BM. For instance, term weights are not allowed. These drawbacks cause the BM to be generally viewed as inferior to the VSM (see the following paragraph) and the probabilistic model ([SJW97]).

Some of these drawbacks have been eased or even lifted by derived non-binary models. For instance, fuzzy set models (see e.g. [Boo80]) and the more general Extended Boolean model ([SFW83]) allow the inclusion of term weights. In the Extended Boolean model, the $p$-norm varies the strictness of the Boolean operators. In the extremes, either the BM ($p = \infty$) or the VSM ($p = 1$) is obtained. According to [Sav94], the Extended Boolean model gains a significant improvement over the BM and the probabilistic scheme. In addition, it is to be favoured over the VSM.

Vector Space Model

The Vector Space Model ([SWY75]) has a geometrical point of view. Query and characterisations are represented as vectors in a multi-dimensional information space. The similarity between vectors is measured by the difference of their direction. More precisely, the size of the angle between two vectors is inversely proportional with their similarity. Vectors pointing in the same direction have high similarity.

The best known system based on the VSM is the SMART system ([SM83]). From the 1960ies on, many experiments have been conducted with the SMART system. Section 3.3.1 illustrates experiments with SMART performed within the Profile project.

**Example 2.1** In Figure 2.3, two example vectors are provided in a three-dimensional information space. In IR, the VSM is applied to information spaces with much higher dimension. Consider as terms WWW, Internet, and computer. The axes in Figure 2.3 denote the number of occurrences of these terms. Suppose that a certain document contains two occurrences of the term computer, a single occurrence of WWW, and does not contain the term Internet. The contents of this document can then be represented by the vector $(2,1,0)$. The query vector in Figure 2.3 denotes that WWW and Internet have weight 2 and that computer is not featured in the query. The angle $\alpha$ between both vectors, as denoted by the dashed line, is a measure of the similarity between both vectors.

The VSM is not fully specified for practical use. For instance, the dimension, i.e. the number of terms used in vectors and the terms to include are unspecified. In addition, there are
many ways to compute weights for query and document terms in vectors. A popular term weighting scheme is the tf*idf (Term Frequency * Inverse Document Frequency) measure. This measure divides the number of occurrences of each term in a document by a value proportional to the number of documents the term occurs in. The tf*idf measure thus denotes the discriminating power of a term. In \[SB88\], automatic term weighting techniques are discussed and compared.

The VSM is a best-match model, meaning that gradations of relevance can be computed. This constitutes an important advantage over the BM. Other advantages are that the VSM has an intuitively appealing notion of similarity, allowing novice users to easily grasp the workings of the system. In addition, it provides a unifying basis for several IR operations. For example, characterisation techniques may be compared by the discrimination values of the indexing terms they yield. Indexing quality is then measured by how well spread the document vectors are put in the information space.

The lack of structuring mechanisms, such as Boolean operators, may be considered as a disadvantage. Furthermore, the VSM assumes that vector terms are orthogonal, that is, the number of occurrences of a term is independent of that of other terms. Clearly, this often is questionable. For instance, terms Internet and WWW will often be encountered together in the same document. In addition, explicitly stating synonymic (OR) and phrasal (AND) relations is difficult in the VSM.

**Probabilistic Models**

Relevance estimates lack certainty due to inherent impreciseness in natural language, the partly specified nature of user interests, and personal biases concerning relevance. This
observation supports the idea that probability appears to be a natural basis for aboutness claims.

The first explicit reference to a probabilistic approach of IR was made by Maron & Kuhns ([MK60]): “The result of a search is an ordered list of those documents which satisfy the request ranked according to their probable relevance.” This is stated as an optimality statement in the well known Probability Ranking Principle (PRP) ([Rob77]): “If a reference retrieval system’s response to each request is a ranking of the documents in the collections in order of decreasing probability of usefulness to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data has been made available to the system for this purpose, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data.”

The Okapi research projects ([Wal89, RWB+96]) have delivered implementations of catalogue systems based on the PRP.

In short, probabilistic IR systems work as follows. Query and document characterisations are represented by sets of terms. Terms are modeled as stochastic variables. Then, the probability that document \( d \) is relevant given query \( q \) and the probability that it is not relevant are computed. Computation of these probabilities involves rules from probability theory. If the probability of relevance is higher than the probability of non-relevance, then the document is rendered.

The probabilistic model has been reported to perform significantly better than the BM, but not as good as the VSM ([Sav94]). Theoretically, the VSM and the probabilistic model can be translated into one another ([TC92]). A disadvantage of the probabilistic model, as for the VSM, is that term independency is to be assumed. In addition, estimating term weights without relevance information is hard. Furthermore, probabilistic tools are so refined that the user cannot understand them ([Rij86]).

**Advanced Logical Models**

Retrieval models can be interpreted to describe forms of logical implication. The logical uncertainty principle, as stated by van Rijsbergen in [Rij86], phrases this as follows: “Given any two sentences \( x \) and \( y \); a measure of the uncertainty of \( y \rightarrow x \) relative to a given data set is determined by the minimal extent to which we have to add information to the data set, to establish the truth of \( y \rightarrow x \).” It should be noted that the symbol \( \rightarrow \), here, not necessarily corresponds to material implication. Rather, some form of plausible inference is aimed at.

Different forms of inference have been considered for IR, each focusing on different issues. There are, for instance, Modal Logic ([Nie89, CN90, Nie92]), Situation Theory ([Lal96]) (combining this theory for information with Dempster-Shafer’s theory for uncertainty), Default Logic ([Hun95]), and Description Logic ([MSST93, Seb94]) (describing structure and content of documents).
As an example advanced logical model, we sketch Preferential Models (PMs) ([KLM90]). PMs for IR ([Bru98, WBHW98b]) deal with non-monotonic reasoning with respect to aboutness decisions. In this thesis, the description of PMs aims at illustrating advanced logical models. A more complete description of our work on PMs can be gathered from [WBHW98b, Won96, WHHW96].

In PMs, Boolean expressions are used as descriptors and the meta operator $\sim_N$ denotes plausible inference. The expression $i \vdash_N j$ denotes that, in the light of information need $N$, one may plausibly derive $j$ from $i$. Expressions of that form are called conditional assertions. The information need at hand thus influences the derivations that are allowed in PMs. This is done by describing the information need in terms of a so called preference relation. This preference relation states which documents are preferred over others. One may plausibly derive $j$ from $i$ if all most preferred documents dealing with topic $i$ are also about $j$. In PMs, a document is considered relevant to a query if some descriptor out of the document characterisation plausibly infers the query.

Depending on properties of the preference relation, certain derivation rules are valid. For instance, if the preference relation is a strict partial order, the following rule is supported by the corresponding PM:

$$i \vdash_N j, i \vdash_N k \quad \Rightarrow \quad (i \land j) \vdash_N k$$

Cautious Monotonicity

It states that the left hand side of a conditional assertion may be augmented with information it already contains. This cautious form of monotonicity aims at preventing unwanted augmentations. In this way, PMs conservatively deal with non-monotonic user preferences.

PMs feature more derivation rules and support a second meta operator: the preferential preclusion. A preferential preclusion $i \perp_N j$ means that the user is not interested in $i$-documents that are also about $j$. In this way, PMs allow for formulation by exclusion on a meta level.

Modeling navigational query formulation in (special) hypertext environments by PMs is described in [Bru98, WHHW96, BL97]. By formulating his query through navigation, the user constructs a search path in the network. Search paths are translated in several conditional assertions, preferential preclusions, and the negation of both. In other words, search paths describe PMs. By derivation, additional formulae may be derived. Essentially, this enables a set of relevant documents to be identified for queries.

How PMs can be extended with domain knowledge is spelled out in [WBHW98b]. For example, equivalence relations and containment relations, i.e. counterparts of logical equivalence and implication, are included in PMs. Synonymy can be seen as an equivalence relation and the is-a relation may be viewed as a containment relation.
Inference Networks

Inference networks (\cite{TC90}) represent a generalised probabilistic model. An advantage of inference networks is that, in ascertaining the degree to which an information need is satisfied, several types of information may play a role. That is, inference networks explicitly allow several types of information to be included and combined as evidence. For example, a number of document representations stemming from different indexing techniques and the interaction between them can be included.

Inference networks are directed acyclic graphs of nodes, representing propositional variables or constants, and links, denoting dependencies between nodes. A link denotes that the parent node implies or causes the child node. The quantitative influence of all parent nodes is expressed in so called link matrices.

Inference networks can be divided into two parts, as sketched in figure 2.4. The upper part, the document network, represents the document collection and only needs to be constructed once. It consists of three types of nodes. The root nodes, called document nodes, \(d_i\), denote (the observation of) documents. Text representation nodes \(t_j\) denote the content of documents. Several representations per document are allowed, making room for inclusion of, for instance, different indexing procedures. Finally, concept representation nodes \(r_k\) denote primitive concepts that appear in the document collection at hand. Although the number of nodes is theoretically unlimited, practical constraints imply that their number in applications will be bounded.

The lower part, the query network, represents dependencies in the information need. The query network is to be rebuilt for each new query. The single leaf \(Q\) denotes the degree to which the information need is satisfied. Multiple queries, represented by nodes \(q_i\), may be used to express the information need. The multiple roots \(c_m\) of the query network express the concepts present in the information need. These nodes are coupled with concept nodes in the document network.

![Figure 2.4: Example Inference Network.](image-url)
Inference networks integrate several existing models into a single framework. Inference networks can express the BM and VSM by instantiating the ways in which term weights are computed (see [TC92] for details).

Natural Language Models

The goal of NLP in supporting IR concerns easy, clear, and context-sensitive query formulation and better retrieval by normalisation of structure, form, and meaning. Two important steps in NLP are syntactic analysis (see e.g. [ATKW98]), after which resembling structures can be mapped onto one another, and lexical analysis, which offers the opportunity to normalise conjugations and compositions.

For the sake of clarity, we view a natural language model as one that takes linguistic structure beyond the single word level into account. That is, descriptors of a natural language model contain linguistic structure in the form of inter-term relations, which must be taken into account during matching.

This includes a whole spectrum of models, according to the amount of linguistic structure that is taken into account. A bag of keywords does not involve linguistic structure, and, thus, is not considered to be a natural language model for IR. Word pairs may be viewed as representing some, albeit very little, structure. As such, they may be thought of as a very simplistic natural language model. Other points in the spectrum are, sorted by increasing complexity, index expressions, noun phrases, and verb phrases (see e.g. [AWKB00b]).

![Diagram](image)

Figure 2.5: General setup of an NLP component in an IR system.

In general, an IR system based on a natural language model involves a number of steps, as depicted in Figure 2.5 ([Str94]). These steps may exploit additional knowledge resources. For instance, word-type lexica are often applied in tagging. In addition, grammars may be exploited to parse text into the right linguistic structure. In addition, lexica may be used for normalisation of descriptors. In some cases, domain models are used to map descriptors to restricted ontologies ([Bak98]).
CHAPTER 2. INFORMATION DISCOVERY

Cognitive Models

The models mentioned so far are focused on query and document representations and matching; they do not consider cognitive and social aspects of IR. This observation is consistent with [Ell92], where a distinction is made between physical and cognitive paradigms for IR. Cognitive models put the user in a central position. The information seeking behaviour of users is the focus as well as the way in which IR systems are used in operational environments. Cognitive models are called holistic in ([SJW97]), indicating that these models pay regard to the “whole” of IR rather than just to parts of it.

Cognitive models deal with, for instance, perception of relevance (see [Miz97] for an overview of the history of relevance), design of user friendly interfaces, interaction and development of information needs (the way the information need may change due to interaction with the system) (see e.g. [BOB82, CR96]), how existing tools and models can be combined to get an effective technique. As examples, consider the often advocated synthesis of hypertext (browsing) and retrieval (searching) (see e.g. [LZ93]), and information seeking behaviour by using problem structures for dialogues ([DBB95]).

Choosing an IR model

Formally, several relations between IR models can be proven ([TC92, SFW83, Hui96, HW98]). These relations include instantiations (e.g. the Extended Boolean model can be instantiated as the vector space model) and different embeddings and equivalences ([Hui96]).

Practically, the suitability of a model depends on the kind of searching it is used for. For example, for known-item searching exact match models perform well ([TC92]). Sophisticated IR techniques such as summary generation, however, require linguistic aids. Furthermore, explaining the matching process can be based on symbolic inferences of logical models.

2.3 Information Filtering

Filtering and retrieval are two approaches to searching information. In many respects, the filter problem can be seen as a dual problem of IR ([BC92]). This section briefly sketches the conformities between IR and IF followed by a description of their differences.

Filtering and retrieval conform at an abstract level. Both approaches have the same goal: providing people with the information they require while minimising the amount of irrelevant documents. In addition, they support the same conceptual task separation into modelling user interests, indexing document content, and matching. The remainder of this section describes the aspects that are especially relevant to filtering: persistency of interests, streams of documents, interaction, and social aspects.
2.3. INFORMATION FILTERING

2.3.1 Persistency of Interests

Queries in IR correspond to momentary information needs which are not supposed nor required to last over a longer period of time. In other words, IR deals with short term interests. Of course, long term interests can be dealt with by IR systems by simply firing a query several times. However, many IR models have no built-in facilities to capitalise on this.

Filtering, on the other hand, considers stable information needs. Thus, for some period of time, the interest does not undergo radical changes. This means that the interests are to be stored persistently, which is done in a so called user profile. In user profiles, also periodic goals and desires can be stored. Thus, IF deals with repeated uses of the system, i.e. with a series of information seeking sessions.

It is inherent in learning by receiving new material, however, that the interest develops over time ([BOB82]). User profiles thus need to be modified. This introduces the need for tools to support profile adaptation.

2.3.2 Stream of Documents

Traditionally, IR has focussed on stable collections of documents. Stating this slightly less strict: IR has dealt with collections having a low modification rate. This means that documents are infrequently modified, added, or removed. Opposed to this, IF deals with a dynamic document stream instead of a stable document collection. Example streams of documents are incoming mail and news messages. A stream of documents can be also obtained by having agents harvest the WWW and select documents based on some criteria.

Documents in a stream can be considered to have a distribution need ([HSB+96]). The characterisation of the document can be seen as a description of its distribution need and is matched against user profiles, i.e. descriptions of the information users. In other words, the initiative is with the documents rather than with users.

In document streams, representations of documents are generally not organised. For stable collections of documents, IR systems may exploit databases and other auxiliary structures formed on the basis of the document characterisations. An auxiliary organisation in the form of a navigational network is discussed in section 4.2.

2.3.3 Interaction

From the stream of documents, the filtering system allows only relevant documents to proceed. Filtering can thus be considered as a process of identifying irrelevant documents, rather than directly selecting relevant material ([BC92]). In this viewpoint, filtering involves the removal of documents from a stream whereas retrieval involves the selection of documents from a collection. This is sketched in Figure 2.6. A well known example of such removal is the deletion of junk mail by e-mail filters.
IR is aware of the possible inadequacy of queries as representations of information interests. As a consequence, interaction during matching is often allowed. This means that, in IR, query formulation is an iterative process during a single information searching session.

In IF, the user profile is assumed to be a correct specification of his information need ([BC92]). This means that the descriptor language used should enable a concise formulation of information requests. The reason for this is lack of interaction during a single session.

### 2.3.4 Social Aspects

IF acknowledges that individual users belong to some community. In this sense, social aspects are exploited in filtering out relevant documents. In social filtering, also called collaborative filtering, users help each other to evaluate what is of most value [RIS+94]. In [GNOT92], a filtering system is described in which users may annotate documents. Based on these annotations, other interested users are searched. The user community can also be modeled by several relations between users. Such relations can be grounded upon situational characteristics, such as the profession of a user and his department, or on topical relations, relating users having similar interests.

Using information about other users for filtering makes privacy an issue of concern. In searching information, it may not always be the case that personal interests are favoured to be shared. The influence of privacy in retrieval and filtering is examined in section 2.5.4. Privacy has not played an important role in IR.

### 2.4 The Information Discovery Paradigm

The traditional paradigms of IR and IF, i.e. modelling a single user and a single source, have clear shortcomings in a networked setting plagued with an information glut. Therefore, a
new paradigm is described which consists of a synthesis of the paradigms for IR and IF. This combination of IR and IF is what we call Information Discovery (ID).

![Information Discovery Paradigm](image)

**Figure 2.7: Information Discovery Paradigm.**

ID features a networked environment of multiple users, sources, and intermediaries called information brokers (see Figure 2.7). The intermediaries facilitate the flow of documents to interested users. This may be done at direct user requests (IR) or by forwarding documents that are presented by some source (IF). Information brokers form the pivots of the ID paradigm. The architecture described in [KGF99] features a similar conceptual division into users, resources, and (special service) brokers for accessing digital libraries.

In general, the entities in ID may communicate freely with each other. A special instantiation is obtained by not considering direct communication between requesters and providers. This conforms to the notion of brokering as described in [DS97]. In this way, privacy of both requesters and providers may be maintained as elaborated in Section 2.5. As sketched in Figure 2.7, two types of communication remain. First, communication between entities of the same type, as represented by the dashed lines. Second, communication between a broker and a different entity, signaled by the solid lines. This restricted form of communication ensures that users and sources communicate via brokers.

### 2.4.1 Benefits of the Synthesis

Integrating IR and IF has several benefits. Combining IR and IF in a single application means the user only has to work with a single system at little or no expense of increased complexity. As such, ID systems provide a uniform interface to a variety of information sources ([HS97]). Moreover, IR and IF can mutually benefit from their synthesis in ID.
These benefits stem from social aspects and the combination of information about users and documents ([WBHW98a]). Section 6.7 describes an application of combining both sorts of information.

Another beneficial aspect of the synthesis of IR and IF is that the user profile forms a naturally personalised context to expand the user query in. The query can be expanded in the context of a single topic of the profile. This is called topic expansion and requires a (set of) topic(s) to be assigned to queries. A form of widened expansion is obtained if the user profiles of related users are taken into account as well. This may also be called social expansion. Social expansion can be refined if communication channels are labeled. Then, only special types of channels can be selected, for instance focusing on employees of the same department.

Furthermore, user profiles may be adapted based on queries. Three moments for this can be identified. First, the moment when a new query is specified. Second, on the rendering of documents to this query. Third, at the stage of relevance feedback. Thus, by exploiting user queries for the adaptation of user profiles, IF benefits from the synthesis with IR.

In a way similar to obtaining efficient brokers by serial composition, efficient filtering can be achieved by serially combining user profiles. Starting, again, in a bottom up fashion, the user profiles are preceded by less specific ones. Matching, then, starts with rather general group profiles, gradually proceeds through more complex group profiles, and, finally, ends in the user profiles.

Synthesising IR and IF also combines their difficulties. For instance, having to cope with dynamics of both worlds, broker design is very important. This is elaborated in section 2.5. The next section discusses automation of ID tasks.

2.4.2 Agent-Based Information Discovery

The dynamic nature of electronic information and the huge numbers of information providers and users call for an automated approach. Therefore, the entities of the ID paradigm are viewed as agents. Agents are software programs that act on behalf of their human users in order to perform recurrent or laborious information searching tasks.

Conform the ID paradigm, we discriminate user agents, broker agents, and source agents. Conceptually, the division into three types of agents is comparable with the computational architecture proposed in [DS97] as general agent-based architecture for problem solving. Together, the agents form a multi-agent system for information disclosure.

In this thesis, focus is on the use of agents for ID rather than directly on the notion of agency. Consequently, we do not consider agent-specific fundamentals such as planning, scheduling, and task structures. For an elaborate description of agents, the reader is referred to [HS97].

Despite of, or maybe due to, the large amount of interest in agent technology, there is little agreement on agency ([Lin96]). That is, many definitions of agency exist, giving different characteristics to agents. Rather than discussing them all, we select a suitable
description for ID that consists of four characteristics ([WJ95]). The usefulness of the chosen description, the so called notion of weak agency, is illustrated below.

**Reactive** In ID, the environment of agents consists of users, communicating with the agents through interfaces, sources of documents, and other agents. Agents perceive their environment and are able to respond to changes that take place in it. This is why agents are reactive.

Clearly, agents in ID should react to search requests from users, e.g. by returning a set of relevant documents. In order to render documents, agents should also contact other agents, introducing the need to react upon requests from other agents.

**Social** Agents interact with the entities that comprise their networked information environment. In ID, for instance, broker agents may combine their forces to offer better matching services. The social ability of agents, e.g., their ability to communicate, requires some kind of agent-communication language. For this topic, the reader is referred to [LF97].

Cooperation in multi-agent systems may involve different organisational structures ([DS97]). An example organisational structure, the economic market, is discussed in [WBHW98c].

**Autonomous** Autonomy of agents means that they have some control over their actions and internal state and may operate without the direct intervention of humans. For instance, [RGK97] describes agents that autonomously navigate a network to gather information.

**Proactive** Next to reacting to changes in its environment, an agent is able to take the initiative and proactively exhibit goal-directed behaviour. In ID, anticipating user interests and preferences by proactively taking the initiative may spare the user in his quest for information.

The current state of the art encountered in search engines does not show general proactiveness. Rather, it shows interval proactiveness by executing periodic actions ([DS97]). Users may set the duration of the interval and, as such, can control periodic actions. In [LPT99], this is provided by so called continual queries where monitoring involves event-driven trigger evaluation.

### 2.4.3 User Agents

By analysing user behaviour, user agents derive user goals, interests, and information needs. This process is called user modelling. In Chapter 3, the user modelling component in the PROFILE project is described. A user agent forms an abstraction of the user, called a user profile, the representation of which is actually worked with. Important research questions for this component, which, however, fall outside the scope of this thesis, are formulating
and adapting user profiles. Each user is appointed a number of user agents, allowing for different views on user behaviour. A generic framework for coherent user-agent interaction is provided in [RS97]. User agents may exploit feedback mechanisms to personalise their assistance ([CH99]). In addition, user agents may collaborate by sharing their knowledge about users to learn faster and broaden their competence ([LMM97]).

A user agent enables the user to specify a query, a description of a short term interest, which belongs to a single information need. In addition, it adds abstractions of the (long term) interests of the user to the user profile. Each long term interest corresponds to a distinct information need.

By finding common interests among a group of users, a group profile can be constructed. A special user agent may be assigned to the group profile, communicating with the individual user agents. Essentially, this allows the construction of a hierarchy of user agents. In this hierarchy, it is important to determine to which group profile, if any, a user’s information need may be assigned.

Autonomous IR is querying on initiative of user agents without direct signal from a user. This is enabled by constructing queries on the basis of the user profile. For autonomous IR, user agents proactively anticipate the user’s information need. In [Kle99], this is implemented by prefetching embedded (anticipated) documents.

2.4.4 Resource Agents

The term information source is used to denote a collection of documents. A resource agent is an agent that has access to a number of information sources. Furthermore, resource agents are able to characterise documents in a particular way. Each information source is accessed by a number of resource agents. This allows for different views on documents, or, complete sources. For example, documents from the same source may be indexed differently by several resource agents. In [BBB+97], resource agents are featured that make information from local sources available for retrieval by integrating ontologies. Resource agents that autonomously travel networked environments to gather information are described in [RGK97].

Resource agents could also structure the representations of the contents of sources, providing an overview. This may yield ancillary layers in the form of lithoids (see section 4.2) or association indices (see section 6.7).

2.4.5 Broker Agents

Broker agents form intermediaries between user agents and document agents. They act as information brokers, providing user agents with relevant documents. Broker agents match document characterisations with user profiles to establish degrees of relevance of documents with respect to user interests. In [SKWJ99], broker agents are described that incorporate several matching strategies. Broker agents may also implicate other brokers in their decisions. This is touched upon below.
In practice, information brokers perform additional tasks next to matching. These include, for instance, monitoring sources (signaling updates or changes in some condition), merging results from several sources by finding a common scale for several aboutness measures, and contacting several user agents when new documents are offered. These tasks illustrate that information brokers act as middle agents ([SLKW99]).

Middle agents form intermediaries between two parties. They can be considered from several conceptual points of view. Example types of middle agents are mediators, black boards, and arbitrators. Several reasons may be advocated for implementing information brokers as middle agents (see e.g. [KH97]). First, rather than relying on traditional techniques that are user-driven, brokering allows sources to contribute to information gathering as well. This is an essential criteria for ID since it features both IR and IF. Second, the imbalance and duality in ID require strong intermediaries. Third, the conceptual separation eases development of specific agents and effectuates a flexible multi-agent system for information disclosure. This is further elaborated in Chapter 3.

The broker's aboutness decisions can be modeled as a qualitative inference process (see Section 2.2.3). The aboutness decisions can also be based on numerical similarity measures. Broker agents can be characterised by the way in which they cooperatively make aboutness decisions. In a network of brokers their aboutness decisions may be derived from the aboutness results of other brokers (see e.g. [HL96]). That is, in making aboutness statements, brokers may consult other brokers. These other brokers are called the contacted brokers.

In [WBHW98a], several types of information broker are defined, based on the influence of the contacted brokers. For instance, a unanimous broker makes aboutness statements if all contacted brokers agree. A lawyer agent, not wanting to miss relevant information, considers a descriptor about another if there is no contacted broker that says the contrary. A form of meta-search, i.e. merging the results of a number of broker agents, is obtained if a typical broker is used. All documents which are considered relevant by at least one contacted broker are rendered by a typical broker. Meta search engines for the Internet may be considered typical brokers.

For reasons of efficiency, a broker agent that supports a competent aboutness relation at high costs, can be preceded by a less restrictive and rather cheap broker. We call this process serial composition, and it can, of course, be repeated several times. In [WBHW98a], the conditions are spelled out to construct series of increasingly complex brokers, so-called broker filter paths. It is also shown how broker filter paths are to be combined with hierarchical user profiles.

2.5 Duality and Broker Design

As described in section 2.4.5, information brokers can be implemented as middle agents in several ways. A major research question is thus which conceptual type of broker to use in ID. Important issues concerning this are how user and document agents locate information brokers, privacy of interest, and generality of the aboutness result.
The focus of this section is on the conceptual design of information brokers in the context of the ID paradigm\(^1\). This is even more important since the dual nature of the IR and IF paradigms yields an imbalance in ID. Our goal is to develop an instrument which explicitly shows the influence of this duality on the conceptual design of information brokers. The instrument provides a solid basis for analysis of duality in ID.

Although the proposed application domain is ID, the instrument is applicable in every paradigm that exhibits a dual nature and uses middle-agents. For instance, middle-agents in electronic commerce (see e.g. [WBHW97b]) can be focused on. This also holds for numerous other applications of WWW agents, and middle-agents in general. As an example, consider BargainBot ([Aou96]), helping users with shopping and IR, and Kasbah ([CM96]), aiding users in buying and selling goods.

This section has the following structure. Section 2.5.1 elaborates on the origin of duality between IR and IF. Section 2.5.2 shows that, within ID, this duality has a cumulative nature. Section 2.5.3 introduces the instrument for analysing the influence of cumulative duality on broker design. Its use is illustrated in section 2.5.4. Finally, section 2.5.5 provides several additional issues concerning cumulative duality in ID.

### 2.5.1 Duality in Information Discovery

Duality in ID stems from the synthesis of the dual paradigms of IR and IF. On a basic level, i.e. the level of contextual assumptions, this is evident in, for instance, dynamic vs. static user needs and sources of information. On a higher level, the goals of users and sources are dual: users want only relevant information to be rendered (their preference is to receive *not too much* information), whereas sources want to deliver as much information as possible (their preference is to distribute *not too little* information).

A result of both forms of duality between users and sources is an imbalance in the ID paradigm. To guarantee fairness in an unbalanced paradigm, neutral intermediaries are introduced: information brokers. As stated in section 2.4.5, the more basic goal of information brokers is the matching of user interests and available documents.

A common aspect of information brokers and Negotiated Retrieval ([NPLL96]) is the merging of documents out of several sources. However, information brokers have more goals and features. Information brokers should, for instance, be proactive to relieve the user as much as possible.

### 2.5.2 Cumulative Duality in Directed Communication

If all communication between users and sources goes through brokers (see Figure 2.7), information brokers are true middle-agents. Middle-agents may be considered as an augmentation of the client-server model (see e.g. [Wie92]). Information brokers, however, are

\(^{1}\)This section is based on [WBW98d].
2.5. DUALITY AND BROKER DESIGN

2-way tasks: users and sources can act both as clients and as servers. In this sense, middle-agents in ID are an extension of a double client-server model.

These 2-way tasks in ID result in a special form of duality, having a cumulative nature. It is cumulative over the directions of communication: either from user, via broker to source, or the other way around. Consider, for example, user privacy. Knowledge about user preferences initially is only available in the user himself and can only reach a source through a broker. The dual criterion, concerning source openness, is cumulatively communicated the other way around. Other instances of cumulative duality are considered in section 3.

2.5.3 Cumulative Duality Matrix

Most specific ID criteria embody a cumulative user-source duality. We propose the Cumulative Duality matrix (CD matrix) as an instrument for representing and analysing the influences of cumulative dual criteria upon the design of information brokers. These criteria should be incorporated in a method for agent design. In, for instance, the AWIC method ([Mue97]) the criteria influence the Interopability and Coordination models.

The general form and characteristics of CD matrices are depicted in Figure 2.8. The cumulative user criterion is set out vertically. Its cumulative nature is expressed by increasing the number of entities that conform to it: from users only, via users and brokers, to users, brokers, and sources. The dual source criterion is set out similarly in horizontal fashion. The nine cells systematically enumerate the possible combinations concerning the dual criteria.

![Figure 2.8: General Cumulative Duality Matrix.](image)

The upper left-most cell in Figure 2.8 represents the most critical situation, since the information broker has no means to deal with neither user nor source criteria. As we go to the right from here, broker and user agents become capable of dealing with the source criterion. Therefore, a user oriented solution is forced, i.e. one that demands most of users.
If we go down from the first cell, the same occurs with respect to the user criterion, i.e. it forces a source oriented solution.

The middle cell is the minimal cell for which the broker can cope with both the user and source criteria. Since no extra capabilities are available in either user or source, this cell demands a strong broker.

In the cell right from the middle cell, the user is also aware of the source criterion. This means that a nuance, relieving the broker’s burden, towards the user can be made. Analogous remarks apply to the cell below the middle cell. The last cell represents the potentially most powerful situation since both user and source can aid or direct the broker.

The above is easily instantiated with, for example, the user criterion regarding dynamics of user interests. The first row then represents that only the user is able to deal with user dynamics. If brokers can deal with user dynamics too, we find ourself in the second row, and if sources are equally capable in the third row.

2.5.4 Instantiations of CD Matrices

This section illustrates the workings of CD matrices with three example criteria. In addition, a number of other cumulative dual criteria are mentioned.

Partial environmental knowledge

Environmental knowledge in ID concerns the locations, names, or addresses of the agents. Information brokers need this information about both users and sources. If users know the names of sources, they can send addressed queries (directed IE). If sources know the names of users they can send information directly (directed IF).

Figure 2.9 provides a CD matrix for the dual location criteria. First we consider the critical situation given in the upper left-most cell. In this situation, the broker initially does not know how to access either users or sources. This means that the broker is passive, i.e. both users and sources have to take the initiative. Processing of queries and incoming documents can only be based on coincidental co-occurrence.

User and source initiative. Next we consider the top row of the matrix in Figure 2.9. In the upper middle cell, the broker can contact sources but not users. This means that users have to take the initiative, leading to a typical IR situation with source querying. In the upper right-most cell, users can perform directed source querying, but still have to take the initiative.

Analogous situations arise in the leftmost column. Here we have sources taking initiative, leading to a typical IF case with user answering. In the bottom cell of the first column, sources have directed user answering on the initiative of the sources themselves.
2.5. DUALITY AND BROKER DESIGN

<table>
<thead>
<tr>
<th>Location of User known by</th>
<th>Location of Source known by</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S+B</td>
</tr>
<tr>
<td>U</td>
<td>Passive Br</td>
</tr>
<tr>
<td></td>
<td>U/S-initiative</td>
</tr>
<tr>
<td></td>
<td>Coincidence Br</td>
</tr>
<tr>
<td>U+B</td>
<td>S-initiative (IF)</td>
</tr>
<tr>
<td></td>
<td>U-answering</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>U+B+S</td>
<td>S-initiative (IF)</td>
</tr>
<tr>
<td></td>
<td>Dir U-answering</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.9: CD Matrix for Partial Environmental Knowledge.

Proactive brokers. In the central cell of Figure 2.9, the broker is the pivot of the system. Since it knows both user and sources, it can perform proactive IR and IF by contacting users and sources on its own initiative. Privacy of location of both users and sources is guaranteed, which also implies that users and sources cannot assist the broker.

In the cell right to the central one, the location of sources is also known by users, which therefore can also perform directed IR. Still, user privacy remains untouched (undirected IF). Analogous remarks apply to the cell below the central one. The location of users is then known by sources but not vice versa, leading to directed IF but undirected IR. The final cell is the only one totally without location privacy, resulting in fully transparent information brokers.

Privacy of interest and content

When searching the Internet, we would often prefer to keep our profile of interest hidden for the outside world. This is an important point in designing information brokers and is to be treated with care. In privacy-sensitive cases, it may be required to keep our search interest private, while in other cases it may help us to make (part of) our interest known to external agents.

In this section we discuss this in the form of two dual criteria. These criteria concern privacy of user interests and source content. The resulting CD Matrix is given in Figure 2.10.

First we consider the upper left-most cell of Figure 2.10. In this most critical case, the information broker has no instances of interest and no documents to match. The only thing left for the broker is to broadcast requests and proposals, and assemble the answers.
**User interests known by**  

<table>
<thead>
<tr>
<th>User interests known by</th>
<th>Source contents known by</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td><strong>S+B</strong></td>
</tr>
<tr>
<td>Naive Br</td>
<td>S-overview</td>
</tr>
<tr>
<td>Broadcaster</td>
<td>Hyperindexing</td>
</tr>
<tr>
<td>Assembler</td>
<td>U-guide</td>
</tr>
<tr>
<td><strong>U</strong></td>
<td></td>
</tr>
<tr>
<td>U-overview</td>
<td>True Broker</td>
</tr>
<tr>
<td>Filter topology</td>
<td>Total privacy</td>
</tr>
<tr>
<td>S-guide</td>
<td></td>
</tr>
<tr>
<td><strong>U+B</strong></td>
<td></td>
</tr>
<tr>
<td>Selective IF</td>
<td>U-recommender</td>
</tr>
<tr>
<td>Narrowed topol.</td>
<td>S-privacy</td>
</tr>
<tr>
<td><strong>U+B+S</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.10:** CD Matrix for privacy of interest and Content.

**User and source overviews.** In the middle cell of the top row, brokers can support an overview of the available information. They can, for instance, support Query by Navigation in a hyperindex (see e.g. [BW92], [Bru93]). In the right cell of the top row, it is possible to perform selective IR, for example by personalising the process of hyperindexing (see e.g. [BB96]). Similar aspects are found in the left column.

**Broker versus arbitrator.** The middle cell represents a true information broker, i.e. the broker is the only one knowing about both interests and available content. Privacy for both users and sources is guaranteed. The cell to the bottom represents the broker as introducer or bodyguard. Since the user is unaware of the sources’ content, it must be introduced by the broker. In the most powerful combination, the information broker can merely act as an arbitrator since users and brokers are mutually omniscient. An example of such a brokering environment is presented in section 6.7. This example is taken from [DSW97] and adapted to ID.

**Dynamical Interest and Content**

Both user interests and available information are dynamic in ID. User interests are communicated cumulatively towards sources. In the dual case, the content of sources is communicated cumulatively towards users. Information brokers form intermediaries between dynamic parties. See Figure 2.11 for the corresponding CD matrix, where the dual criteria concern the capability of agents to deal with user and source dynamics.

**Delayed Interest and Content.** In the most critical case, the broker is a static intermediary. This, of course, is no valid solution. Towards a forced source solution, more parties find themselves capable of dealing with user dynamics but not with source dynamics.
2.5. DUALITY AND BROKER DESIGN

<table>
<thead>
<tr>
<th>Can deal with user dynamics</th>
<th>Can deal with source dynamics</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Static Intermediary</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>U+B</td>
<td>Delayed Doc Provider</td>
</tr>
<tr>
<td></td>
<td>Static Source</td>
</tr>
<tr>
<td>U+B+S</td>
<td>Source Stimulated Filter</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.11: Cumulative Duality Matrix for Dynamical Interest and Content.

Eventually, a source stimulated filter is obtained. Here, the source has to provide snapshots satisfying changing user needs since both users and brokers are incapable of doing so.

**Interface vs. Assistant.** In the middle cell, the information broker is fully responsible for the dynamics of both users and sources. It forms a dynamic interface between users and sources. The cell to the bottom of it represents a dynamic user guide, providing dynamic user access to sources. In the most powerful combination, the information broker can act as a mere assistant providing dynamic mutual support.

2.5.5 Other Cumulative Dual Criteria

Dual time constraints play an important role in ID. First, in IR, users have short term interests and need direct results, whereas in IF, users have long term interests and do not require results to be delivered immediately. Second, the timeliness of documents is treated differently as well. Where in IR the complete set of documents, including old ones, is queried, IF requires relatively new documents to be sent to users.

Traditionally, a user query or profile only consists of a description of the topic of interest. According to Cooper ([Coo71]) not only topicality (logical relevance) but also situational factors (usability) should be taken into account. Dual criteria are obtained with respect to user interests formulated in terms of situational factors as well, and document characterisations supporting these too. The inclusion of situational factors opens the possibility for more interaction in the form of negotiation ([WBHW98c]).

Agents in the ID paradigm form an open environment: agents can come and go when they want. This requires highly dynamic cooperation strategies without strict commitment ([SL95]). The electronic (ID) market stresses the importance of supporting open environment negotiation by letting only the best adapted brokers survive.
Even more cumulative dual criteria can be formulated within ID but also more in general. The consequences of those criteria can be clearly depicted and analysed through the use of CD matrices since these give a structured overview on a conceptual level. This will make the process of broker design better structured and motivated.

2.5.6 Reflection

We introduced Cumulative Duality matrices as a mechanism to analyse the role of the information broker within the ID paradigm, and of middle-agents in dual paradigms in general. We argued that the duality in goals of users and brokers implies the need for neutral intermediaries.

Design criteria for information brokers were shown to exhibit a cumulative dual nature in Information Discovery. The sections above show the influence of a number of dual ID criteria on the conceptual design of information brokers. A number of general conclusions can be made.

First, it appears that the information broker must be able to deal with both user and source criteria to obtain a viable broker design. This means that there are only four possible cells, i.e. those in the bottom right corner.

Second, different points of view can be adopted, resulting in different favourable broker designs and focusing on different regions of the CD matrices. The function of the information broker is to provide a transparent connection between different capabilities, needs, and knowledge. In that way the broker has to ensure a basis of equality between users and sources. Two major points of view are (1) user oriented, focusing on the cell right to the middle, and (2) source oriented, focusing on the cell below the middle.

Third, the potentially most powerful cell is in certain respects the most problematic as well. For instance, user privacy is not viable.

Finally, the dual nature of the ID criteria is visible in CD matrices as well. Taking the diagonal from top left to bottom right as an axis, the left bottom corner and right upper corner are mirror images with respect to the dual criteria used.

2.6 Outlook

Networked environments have established a prominent position as sources of information. This has been an incentive for the fields of IR and IF to focus more on distributed issues. In our opinion, the integration of both fields, such as modeled by the ID paradigm, will form the basis for many future information systems.

With the growing interest in Electronic Commerce (EC) (see e.g. [Zwa96]), new technology is developed which may also serve ID. To bridge the gap between EC and ID, [WBHW98c] describes several cost models for ID. It will be interesting to investigate how these cost
models enable issues and techniques from EC, such as negotiation, advertising, and the formation of cartels, to be applied in ID.

Furthermore, the rising number of agents on the Internet effects an increasingly pluriform agent society. This increases the interest to investigate the possible influence of agents with non-friendly properties. For instance, it should be researched how non-benevolent agents, i.e. agents that do not always do what is asked for ([WJ95]), influence the quest for information. Another example is the notion of veracity, i.e. not to communicate false information deliberately.

Further research concerning cumulative duality in ID can be focussed on a number of aspects of CD matrices. For instance, the combination of CD matrices can be researched in the light of more traditional matrices. The analogy of multiplying and adding (mathematical) matrices may form a starting point. If successful, this leads to the possibility to investigate and analyse hybrid brokers. In addition, CD matrices can be incorporated in agent design tools. For instance, they could be included in the AWIC method as described in [Mue97]. Finally, the use of CD matrices for personalising brokers can be investigated. For this, we envisage CD matrices as part of advanced user interfaces in which the user can select the most appropriate broker variant.
Chapter 3

PROFILE - A Multi-Disciplinary Approach to ID

3.1 Introduction

The PROFILE project, started in 1996, aims to decrease the problems caused by information overload on the World Wide Web. The main goal of the PROFILE project is to develop an effective agent as intermediary between information sources and information users in the context of the WWW. The effectiveness of the PROFILE intermediary is to be enlarged by going beyond keyword-based approaches. The term Information Discovery (ID) was introduced in Chapter 2 to describe the synthesis of document-triggered (filtering) and user-triggered (retrieval) information interchange. Both forms of information interchange are to be supported by the PROFILE system.

Two research groups of the University of Nijmegen participate in the PROFILE project: the Cognitive Ergonomics group of the NICI (Nijmegen Institute for Cognition and Information) and the IRIS (Information Retrieval and Information Systems) group of the subfaculty of Computer Science. The researchers in the two groups each have a different background. The PROFILE project thus integrates several viewpoints from multiple disciplines and organisations.

This chapter sets out to do three things. First, it evaluates the suitability of an agent-based architecture for the PROFILE project. The organisation of the PROFILE project - decentralised research done by groups with different styles of working and cultures - calls for a flexible architecture. We argue that an agent-based architecture supports the required extensibility. However, within PROFILE, agents are not viewed as a central research issue but as a tool to implement and rationalise our work.

Second, this chapter provides an overview of the research done in the different components of the PROFILE project. The PROFILE project divided the research themes over several

\footnote{This chapter is directly based on [SAW+00].}
components. The aims of these components are described. Their results reveal that different levels of progress were realised.

Third, it describes the integration of the research that has led to the design of a prototype system. Issues concerning the implementation of the prototype are elaborated. In addition, the types of agents used in the PROFILE system are described. Furthermore, the workings of the prototype are illustrated.

This chapter has the following structure. Section 3.2 advocates an agent-based architecture for the PROFILE system. Furthermore, it describes the basic setup of the PROFILE project as a cooperation between four components. Section 3.3 describes the research done in each of the components. Section 3.4 describes the prototype, integrating the individual findings. Section 3.5 provides concluding remarks and suggestions for further research.

### 3.2 PROFILE Requirements

In this section, we first examine constraints on the design of the PROFILE system. Then, we advocate that these constraints suggest an agent-based approach. Finally, we show that functionally decomposing the PROFILE system results in a multi-agent system.

#### 3.2.1 Constraints on PROFILE

The following constraints and issues of concern arise from the nature and goals of the project and the intended use of the system:

- The project is essentially a research project. Therefore, provisions should be available to cope with differences in progress of the project parts. It also implies a wish of every part to be constrained as little as possible by other parts.

- The PROFILE system consists of several components. The general required proactivity of the PROFILE system implies that several parts of the system have to be able to take the initiative.

- The multi-disciplinary project is carried out by different organisations with different styles of working and different research cultures. This should be considered when integrating the PROFILE system.

- Different parts of the project use different implementation tools. This calls for special attention on integration.

- The system is intended to work partly on the user's work station and partly on a central server which proactively searches on behalf of different users. PROFILE thus aims at a flexible distributed system that goes beyond a simple client-server model.
• The dynamics of the information environment call for common communication protocols. The shared protocols should be general enough to allow components to react differently to similar requests in different contexts.

• The system should be easily extensible, enabling different instantiations of functionality to be readily incorporated.

These constraints call for a flexible architecture consisting of several independent modules that cooperatively implement an ID system.

3.2.2 Properties of Agents

An agent-based approach addresses many of the constraints described in the previous subsection. Especially, the following characteristics of an agent oriented paradigm are useful:

**Autonomy.** Autonomous agents, to a certain extent, have control over their internal state and planning of their actions. Implementing the project components as autonomous agents alleviates their interdependencies. This prevents general delay of the project if a single part encounters problems.

**Elaborate Communication.** Communication between components is directed towards a flexible form of cooperation. Agents can negotiate and form teams to adjust to the needs of a situation. This suggests a more powerful approach to communication than simple function invocation. Because the precise functionality of each component was not known at the start of the project, the communication protocol between the different components had to be sufficiently general. The protocol prescribes how components should react to the different kinds of messages.

**Proactiveness.** This property of agents is valuable when designing a proactive information filter. The proactivity of the system can be achieved by the proactivity of its agents. Different forms of proactivity can be identified. In a simple form of proactivity, for example, filtering systems periodically render new documents. Ultimately, proactive agents take the initiative without external incentive. The agents used in PROFILE fall somewhere in between.

**Concurrent processing.** The dynamic and distributed nature of the PROFILE system introduces asynchronous events. Synchronous flow of control thus is impractical.

Many other properties have been assigned to agents (see e.g. [WJ95]). However, the goal of this project is not to elaborate on the notion of agency, but merely to use valuable properties of agents in an information seeking environment.
3.2.3 Overall System Design

The multi-disciplinary nature of the system calls for a functional decomposition. The original design has identified four different components for a proactive information filtering task. Each component has been implemented as an agent, having the abovementioned characteristics. Therefore, the PROFILE system essentially is designed as a multi-agent system. As an extra advantage, a multi-agent system goes beyond the client-server model by allowing the decision about where to do processing to be postponed. In addition, multi-agent systems are easily extensible. One can create different agents for the same task and experiment with it. Extensibility is of great importance in open environments like the Web. The following four components were identified:

**User Interaction.** This part delivers the tools for the users to formulate their information need and to react upon the delivery of documents.

**User Modeling.** This part derives representations of information needs and users, and constructs optimised queries or profiles.

**Retrieval.** The retrieval component is responsible for comparing documents with user interests. Matching is thus an essential topic for the retrieval component. Descriptor languages, used to represent user interests and document contents, are an important issue in retrieval.

**Language Processing.** This part of the project analyses documents in order to deliver a representation of their contents.

![Conceptual PROFILE setup](image)

Figure 3.1: Conceptual PROFILE setup. Arrows denote the flow of information.

Every part of the project aims at delivering an agent for the specific subtask. The flow of information between the components is depicted in Figure 3.1. The user interacts with the PROFILE system in order to specify his information needs. Relevant information concerning the formulation process is sent to the user modelling module. The user modelling component maintains a representation of the users’ interests. This is done by observing the actions of the user or by direct interaction. The user module delivers optimised queries to the matching component. The language processing module indexes available documents. Representations
of user interests and document contents are passed on to the retrieval component, which establishes relevance estimates in order to distinguish the set of documents to be rendered. Rendered documents are presented to the user, who optionally gives feedback on the results. The feedback is used to adjust the user model and starts a new information cycle in the PROFILE system. The four major components of the PROFILE project are discussed more elaborately in the next section.

Cooperation between agents receives a lot of attention in scientific research (see e.g. [HS97]). For instance, negotiation is studied to enhance flexibility and effectiveness of multi-agent interaction. In ideal multi-agent systems, advanced cooperation by negation is important. However, in the PROFILE project, the focus was not directly on agent oriented research. Therefore, other issues concerning information discovery have a more prominent place.

### 3.3 Research in PROFILE

Each component of the PROFILE project has its own field of interest, focusing on a subtask of the system. This section provides an overview of these fields. This thesis is concerned with only one of these fields. For detailed information about other fields, the reader is referred to the work of the respective author. It should be noted that the topics of interests are not always strictly divided over the components.

#### 3.3.1 User Modeling

The user modelling component focuses on an investigation of the use of domain knowledge in the formulation of a query. This is not a new idea. Domain knowledge is applied in document indexing (e.g. [BBM+97] and [Sta96]), matching (e.g. [SQ96]), and query formulation (e.g. [Voo94]). Another knowledge based approach is using a semantic network in which nodes stand for words ([ACM91]).

The research in this component deviates from other applications of domain knowledge in query augmentation in that it investigates whether the richness of knowledge representation can influence the success of query expansions. The idea is to use domain knowledge represented in so called ontologies ([Gru93]). These knowledge structures usually represent their domain in a taxonomy of concepts with each concept represented as a frame with slots, slot values, and restrictions on these slot values. More elaborate knowledge structures allow for more different types of expansions. More different expansions enable more possible ways to improve a query. Therefore, our hypothesis is that elaborate knowledge structures can improve ID performance. At the time of writing, this hypothesis has not been fully evaluated. The work on the user modeling component is expected to answer this question in the PROFILE setting shortly.

ID with domain knowledge represented in an elaborate knowledge representation language poses two separate problems. A first question is how an agent that uses this kind of domain
knowledge can be realised using current technology. The second question is how domain knowledge should be applied to improve a query. The following two sections sketch the research that is done to answer these questions.

A Knowledge-Based Query Formulation Agent

A common-sense criticism against a knowledge-based approach to ID is that it would cost too much resources to provide an agent with a new knowledge base in order to do information filtering. Therefore, in order to turn this idea into a useful technology, the agent should be able to re-use existing knowledge bases. One attempt to facilitate the re-use of knowledge has been made by [RFPG96]. Their solution comprises an editor which can be accessed over the WWW. A library of functions and objects enables programs to access these ontologies in a uniform way. An OKBC\(^2\) client can access OKBC-servers on the Internet to query ontologies about concepts, slots, and values or facets on those slots ([CFF+98]).

Knowledge based agents in PROFILE are implemented in an applet in which a user can select an information need represented in ONTOLINGUA ([Gru93]), expand it to add concepts, translate these concepts to synonym sets in WORDNET ([Mil95]), and finally, use expansions in WORDNET to create the final query. The user can interact with the agent at three levels. The current implementation of the knowledge-based agent only expands queries when the user asks for it. The applet extracts its domain knowledge from a central server using OKBC. A socket interface to WORDNET has been developed to acquire lexical knowledge.

Query Expansion Research

The prototype described in the previous section turned out to have some disadvantages. First, the communication layer between the knowledge base server and the user agent was quite complex. This made the construction of new strategies quite tedious. Second, ONTOLINGUA was not written with the purpose to perform inferences. Therefore, no good inference system has been written in ONTOLINGUA. However, inferences are necessary in order to make use of the more elaborate knowledge represented in this formalism.

For these reasons the research upon domain based query augmentation was done using LOOM ([Mac91]) and a socket based interface to SMART ([Sal71]). The JFACC ontology\(^3\) ([VRMS99]) was used as domain knowledge for an experiment. Representations of ten information needs in JFACC concepts were constructed for this experiment. A collection of 1500 documents was gathered from websites that contained information specific to that domain. The non-expanded query was used to collect 100 documents for each query. The documents were scored manually. Then, each query was expanded using different strategies

\(^2\)Open Knowledge Base Connectivity Protocol  
\(^3\)Joint Force Air Campaign Commander
that used different aspects of the knowledge representation. For example, a strategy that
adds hyponyms of a concept views the ontology as a pure taxonomy while a strategy that
expands on slots and slot values uses more of the power of the knowledge representation
formalism.

Figure 3.2 shows how each run was carried out. Every information need was altered by
each strategy to produce a set of queries. Each of these queries was then used to retrieve
twenty documents.

Figure 3.2: A general sketch of how a run is carried out for a single query and a single
information need. Each possible route though the figure delivers one result.

The results suggest that the answer to when and how domain knowledge can help in ex­
 panding a query is not an easy one. For example, the concepts in the ontology turned out
not to be as richly interconnected as was originally assumed. A lot of the concepts could
not be expanded by strategies that depended on connections beyond a taxonomy. A set of
experiments that try to use more of the structure available in an ontology is being prepared
to shed more light on this issue.

3.3.2 Language Processing

The Language Processing component is responsible for providing representations for textual
information objects (documents), a process widely known as indexing. Indexing in PRO­FILE
goes beyond the use of simple keywords to characterise documents. In an attempt to capture
more of documents’ content, we employ natural language processors and linguistic resources.

Initial small-scale experiments with the IRENA system ([ATK97, AT96]) showed the promis­
ing aspects of lexical and morphological expansion of keywords in improving the effective­
ness of retrieval systems. Furthermore, the Noun Phrase Co-occurrence criterion, i.e. co­
ocurrence of keywords or their synonyms or morphological variants of them in the same
noun phrase, was applied successfully in determining whether keywords are semantically related in a more beneficial way for precision than proximity search. Although low recall was obtained, at any rate, the Noun Phrase Co-occurrence criterion can be used for relevance feedback.

In a further investigation, we have extended the traditional Keyword Retrieval Hypothesis to a Phrase Retrieval Hypothesis, upon which we have built a linguistically-motivated indexing scheme ([ATKW98, AWKB00b]). Two types of phrases have been considered for indexing. These are: the noun phrase including its modifiers, and the verb phrase including its subject, object and other complements.

![Phrase Retrieval Hypothesis](image)

Figure 3.3: Light parsing for indexing purposes.

We have defined an abstract representation of these phrases suitable for indexing. Full linguistic parse-trees contain too much linguistic detail, most of which is unnecessary for indexing, as such details reflect mostly the syntactic description of the natural language used rather than the intended meaning. Therefore, we have settled for less than full linguistic parsing, eliminating structures which can be assumed not to be beneficial to indexing. Figure 3.3 gives an example of the detail-level of our syntax analysis: arguably lighter analysis than full parsing while a reasonable amount of structural information is still retained.

![Linguistic Normalisation](image)

Figure 3.4: Linguistic normalisation.

Although phrases can be used in their literal form as terms, the performance is expected to be inferior to that of keywords. On the one hand, phrases achieve better precision, but on the other hand recall will be too low, because the probability of a phrase re-occurring literally is too low. To deal with this sparsity of phrasal terms, we introduce linguistic...
normalisation. The goal of normalisation is to map alternative formulations of (syntactic) meaning to a normalised form called phrase frame (Figure 3.4).

We distinguish three types of normalisation: morphological, syntactical, and lexico-semantic normalisation. Morphological normalisation has traditionally been performed by means of stemming (non-linguistic suffix stripping). Taking into account the linguistic context, we follow a more conservative approach called lemmatisation which prevents many errors of stemming. Lemmatisation can be seen as part-of-speech-directed stemming. It reduces verb forms to the infinitive, inflected forms of nouns to the nominative singular, and comparatives and superlatives of gradable adjectives to their absolute form.

Syntactical normalisation is based on the linguistic principle of headedness: any phrase has a single head. Thus any phrase can be transformed to a canonical form (an ordered relation between its components): head first, followed by its modifiers. The head gives the central concept of the phrase and modifiers serve to make it more precise. Consequently, the head may be used as an abstraction of the phrase, loosing precision but gaining recall. Heads and modifiers may recursively contain phrases. A number of such syntactical transformations from the phrase domain to head-modifier domain have been investigated.

Lexico-semantic normalisation is based on certain relations which can be found between the meaning of individual words, such as synonymy, antonymy, is-a, and part-of. Our approach combines thesaural information from WordNet with statistical word co-occurrence data to establish such word relations. Two possibilities are explored for lexico-semantic normalisation. The first is lexico-semantic clustering which reduces closely related words to one word cluster, and the second is fuzzy matching which introduces a semantical similarity function between words into the retrieval function.

Parts of this linguistically-motivated indexing model are still under investigation, tuning, and evaluation. However, initial experiments have yielded promising results, suggesting that we are not far from finalising a model which should help to overcome the known and long-survived problems of bag-of-words representations.

3.3.3 Retrieval

The retrieval component, also called matching component, is responsible for a comparison between document contents and user interests. Therefore, an important focus in the retrieval component are metrics for expressing the similarity between descriptors. This yields relevance estimates for documents with respect to user queries and profiles, allowing the distinction between more and less relevant documents.

The language of index expressions ([Bru90, BW91]) is considered in the retrieval module. 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. Index expressions feature a simple linguistically motivated refinement mechanism, sometimes referred to as headedness or concept refinement. An advantage of index expressions is that they support the construction of networks that are
suitable for navigational formulation techniques. This allows users to formulate their information need by stepwise refinement in a navigational network. A formal basis for index expressions is provided in Chapter 4. In addition, several similarity measures for index expressions have been devised, focusing on different properties relating to subexpressions. These have been evaluated in the context of navigational networks for index expressions. As reported on in Chapter 6, a dynamic retrieval system has been built that supports navigational query formulation for searches on the WWW.

Special attention is given to a tractable approximation of noun phrases, coined Boolean index expressions (BIEs). BIEs are elaborately described in Chapter 5. BIEs can be constructed by the refinement mechanisms from index expressions and, by inclusion of Boolean operators, with logical concept building. An example BIE is given in Figure 3.5. Next to sufficiently expressive, BIEs are tractable and compact.

![Figure 3.5: Example BIE.](image)

From a user oriented point of view, compactness is beneficial in formulating the information need. Compactness of BIEs is achieved by allowing nested logical operators. This effectuates subexpression sharing. If an information need involves several related concepts, subexpression sharing saves space (for representing the need) and time (for users to read the representation). The nature of the compactness of BIEs has been studied, revealing that exponential compactness can be reached. This means that, compared to index expressions, BIEs offer a very compact representation of complex information needs.

![Figure 3.6: General setup of propositional form.](image)

There are many different BIEs that express the same information. A normalisation function for BIEs has been devised to reduce the syntactical variety yielded by Boolean operators. Normalisation consists of zipping the dyadic operators, providing BIEs in so called propositional form (see Figure 3.6). This form consists of a logical part, containing the dyadic
operators, and an atomic part. Advantages of BIEs in such form are that operations on them can be specified in terms of (slightly modified) functions on classical index expressions. For example, matching BIEs applies similarity functions for index expressions after normalisation.

In order to assist information searchers during formulation of their information need, two tools for constructing and adapting BIEs are described. The first tool combines direct manipulation of BIEs with navigational formulation for classical index expressions. Index expressions that are encountered during navigation may be included in the query or profile at hand and can be manipulated by direct actions. The second tool shows how relevance feedback can be incorporated with BIEs. This tool guarantees good control over the form of the constructed BIE by separately modelling positive and negative feedback.

Combining filtering and retrieval in a single paradigm, as aimed at in the PROFILE project, may result in an imbalanced setting. In order to guarantee fairness in ID, information brokers need to be designed carefully. In the retrieval module, the imbalance was found to stem from the dual nature of retrieval and filtering (as discussed in Section 2.5). Since this duality was examined within cumulative communication, i.e. communication that was required to go via brokers, it is coined cumulative duality. An instrument was designed to explicitly state the influence of criteria concerning cumulative duality on the required role of information brokers. Example criteria concern privacy of user interests, partial knowledge about broker services, and the dynamics of user interests and offered information.

### 3.3.4 User Interface

The user interface is an important component of an information filtering system. This is due to (a) the real-time character of such an interface, (b) the convergence of input and output streams at a single point in time and space, (c) the required degree of user control over the internal computational modules of the information filtering system, and (d) the large design space as regards the information-rendering methods for a wide range of target platforms.

The user interface will eventually be realised taking a number of requirements into account. For an easy development, interface components will be designed according to the characteristics of the chosen multiple-agent architecture. The complexity of interface requires, e.g., that a distinction be made between rendering functions and query-related functions.

Ideally, in an information filtering context, the user interface will also contain a dedicated user agent, which is capable of maintaining dialogs with the user for the specification of user profiles. By necessity, such a user agent requires knowledge representations which are focused on (a) real-world semantics, (b) self-capability descriptors concerning the system’s beliefs as regards its intrinsic information filtering functionality, and (c) pragmatics of search in large real-life digital information sources.

What has been realised until today is of a far more modest nature. A collection of JAVA routines has been developed which allow for the basic functionality required for information
filtering: user enrollment, specification of interest profiles using keywords, and the presentation of dynamic HTML pages using JAVA. However, this preliminary setup of a user interface is rather basic and is currently being evaluated from the point of view of current user-interface guidelines in cognitive ergonomics.

Preliminary results indicate that the user interface in an information filtering context introduces a number of new concepts which will be unfamiliar to the users of the common interactive search engines as these are available on the World Wide Web today. A number of user-interface metaphors have been proposed in other groups, such as “sending out a dog on a hunt for information” or “the personalized and dynamic newspaper”. However, a generally acceptable solution remains to be realised at the user interface level of any information filtering system.

3.4 Current Prototype

This section describes the current implementation which is based upon the research described in the previous chapter. First, we motivate our choice for the multi-agent architecture implementation. Then, we describe the different agents in the PROFILE system and sketch an example to illustrate a possible run with the current prototype.

3.4.1 Architecture

We have derived the following implementation-oriented constraints on our agents from the project goals as described in section 3.2. First, they should be able to run everywhere. Second, the agents do not need to know where every other agent is localised. Finally, the architecture should be robust when a connection to one agent is not available.

For these reasons, the JatLITE ([Jeo97]) architecture was chosen. It implements agents as JAVA threads, located in different virtual machines. Communication exploits KQML ([FF94]) strings sent through TCP sockets. A central router receives messages from every agent and redirects them to the target agent. PNL extends KQML in order to specify information needs, documents representations, and user evaluations.

3.4.2 Agents in the Implementation

For the implementation, our original division into four different agents had to be refined. The implementation involves seven different agent types. First, the user model part was divided into a part that stores the user preferences, and a part that actually delivers the optimised query. Second, proactive searching on the Web suggested a distinction between a module that actively searches the Web to deliver a preselection of potentially relevant documents and a module that indexes these documents. Table 3.1 summarises the different agent types. These correspond to the ones in Figure 3.7. Different instantiations
exist for the Document Collector. Each instantiation creates an initial document stream from a broad query to a search engine. The current prototype may contact HOTBOT or ALTAVISTA. All agents are embedded in the JATLITE framework.

3.4. CURRENT PROTOTYPE


Figure 3.7: Intended cooperation between the agents in PROFILE for the case of interactive query processing. Arrows denote possible communication.

3.4.3 Working of the Implementation

The agents in the PROFILE system cooperatively realise the three main tasks:

**Query Processing.** When a user specifies a query in the user interface, the resulting documents are retrieved by the PROFILE system. The flow of control is shown in Figure 3.7. The Scheduler gets the initial request (1, 2) via the User Interface. It asks additional user information (3, 4) to the Database, optionally expands (5, 6), retrieves an initial set of documents (7, 9) and sends them to the Matcher for filtering (10). The Matcher forwards documents to the Language Processor (11), who parses them and returns the corresponding characterisations (12). The Matcher then performs the matching and delivers the document to the database if the similarity value is above the threshold defined in the information need (13). After putting it in the Database, the document can be rendered to the user (14, 15).

**Document Filtering.** Upon arrival of new documents, the Scheduler contacts the Database to obtain information need representations. It then forwards these to the matcher. Filtering then proceeds as query processing.

**Proactive Filtering.** A stream of documents is generated upon an initiative of the Scheduler who sends an initial broad query. Proactive filtering then proceeds as document filtering.
<table>
<thead>
<tr>
<th>Agent</th>
<th>Function</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface</td>
<td>Interacts with user.</td>
<td>HTML with JAVA</td>
</tr>
<tr>
<td>Database</td>
<td>Stores user characteristics &amp; interests.</td>
<td>JAVA</td>
</tr>
<tr>
<td>Document Collector</td>
<td>Generates a stream of initial documents.</td>
<td>JAVA. Sends a query to several search engines and retrieves the results.</td>
</tr>
<tr>
<td>Matcher</td>
<td>Accepts an indexed document and a query in the form of index expressions and compares each of them.</td>
<td>JAVA, equipped with a C wrapper. Starts up a CLEAN ([PE98]) program.</td>
</tr>
<tr>
<td>Language Processor</td>
<td>Creates index expressions from documents using part of speech tagging and shallow parsing.</td>
<td>JAVA. Starts Brill's POS tagger [Bri94] and a PERL program.</td>
</tr>
<tr>
<td>Expander</td>
<td>Accepts an information need and expands it when possible.</td>
<td>JAVA</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Directs the flow of control and proactivity of the system.</td>
<td>JAVA</td>
</tr>
</tbody>
</table>

Table 3.1: PROFILE agents with their capabilities and tasks.

The PROFILE prototype forms a testbed for new results and techniques of the components in our project. Therefore, the prototype is not a final product, but is constantly under (re)construction when new material becomes available.

### 3.5 Outlook

In this chapter, we have presented the work done in the PROFILE project. The multidisciplinary nature of the PROFILE project introduces the need for a flexible attunement between the components. The agent-based architecture provides a means to capitalise on any overlap between the components. If we had used a more rigid architecture, the project as a whole would have suffered more from the delay in a single component.

The user modelling, language processing, and matching components conducted research in descriptors that goes beyond the keyword-based approach. This means that PROFILE is capable of extracting structured descriptors from text, normalising these descriptors with respect to morphology, syntax, and lexico-semantic issues. In addition, descriptors are created based on knowledge representations and the user model. Furthermore, similarity functions to match structured descriptors have been designed.

The PROFILE prototype performs proactive filtering. The Scheduler adjusts the flow of control to the type of agent showing proactive behaviour. Currently, the Scheduler may start an information cycle proactively by itself or react to proactive calls from a Document Collector or the Expander.

Every research line in the PROFILE system offers opportunities for further research. An interface that accepts information needs in different forms would greatly enhance the appeal
of the PROFILE prototype. Expansions on information need representations can be more elaborate and focus on words that should not occur in a document. The expanded result should yield Boolean index expressions. More complex document characterisations may be the result of more sophisticated parsing techniques.

An interesting consequence of these investigations for the PROFILE system would be that different instantiations will appear for every module. This necessitates a need for negotiation strategies and learning of capabilities, something which was not marked as a research issue for PROFILE in the original line of investigation.
Chapter 4

Index Expressions

4.1 Introduction

Going beyond the bag-of-words model can be done by including structure in descriptors. For example, the Boolean keyword language models logical relations between terms and index expressions (IEs) capture some of the linguistic structure of natural languages, providing a useful simplification of noun-phrases.

Our concern is the theory of IEs, not the linguistic analysis of text. This means that our focus is on defining, manipulating, and comparing IEs. In addition, we introduce and compare different representation mechanisms for IEs. We thus leave the indexing of text with IEs mainly to the NLP component of the PROFILE project.

This chapter has the following structure. In section 4.2, an overview of the state of the art is given. This allows us to motivate our research into IEs. Section 4.3 shows how IEs are built by nesting subexpressions. Based on defoliation, the structural representation of IEs is described. Section 4.4 compares the structural representation with two other formalisms. Section 4.5 formally derives several notions of subexpressions. Sections 4.3, 4.4, and 4.5 are based on [WBW00]. Section 4.6, based on [WBWb], elaborates on similarity measures for IEs. Section 4.7 provides some remarks on additional issues concerning IEs. Finally, Section 4.8 gives an outlook on further research into IEs.

4.2 Fundamentals of Index Expressions

Index expressions (see e.g. [BW92, Bru93]) have proven of value for ID. As simple yet powerful descriptors, they have been applied for query formulation ([Bru90]), document indexing ([BW91]), matching ([BI94, BG94, Won97, Ber98]) and the construction of ancillary layers for hypertext systems ([BW90]).

Index expressions are based on terms and connectors. Terms denote keywords, concept names, adjectives, gerunds, and attribute values. Connectors denote relations between
terms in the form of prepositions (showing place, position, time, or method) and some present participles (e.g. “using”, “having”, and “being”).

Connectors can be labeled by their relation type ([Far80a, Far80b]). For instance, connector of can denote possession, as in castle of queen or action-object, as in removal of leaf. Other relation types are action-agent, position, association, and equivalence.

The general form of index expressions consists of a head or lead term which has several nested expressions that are connected to the head through connectors. The nesting operator \( \otimes \) denotes the nesting of expressions under the head. The notational convention for the nesting operator is illustrated below.

\[
h \otimes_{i=1}^l c_i(I_i) = hc_1(I_1) \ldots c_l(I_l)
\]

Index expressions can be graphically represented as trees. The root of the tree coincides with the head of the IE. The branches, which are labeled by connectors, connect the head to its subexpressions. These, in turn, may be represented as trees. This is schematically depicted in Figure 4.1.

![Figure 4.1: Structural form of index expressions.](image)

In alpha-numerical representation, brackets can be used to denote the intended nesting structure. The following example illustrates different structures.

**Example 4.1** Consider as terms conference, Latvia, and IT. Furthermore, consider as connectors in and on. Consider the following example index expressions:

- conference in (Latvia) on (IT)
- conference on (IT) in (Latvia)
- conference on (IT in (Latvia))

The first index expression has term conference as head and contains two nested expressions, Latvia and IT, respectively. These are respectively attached to the head by connectors in and on. The second index expression has the same head and the same nested expressions.
However, the order of nesting of the expressions is different. The third index expression contains the same terms and connectors as the previous two, but shows a different nesting structure. Its head conference has IT in Latvia as single nested expression, which has IT as head and Latvia as single recursively nested expression.

The empty index expression, denoted by ε, is a special IE. There is a principal difference between descriptors that carry information and descriptors that represent the absence of information. In formulation, the empty index expression can be used to denote an unspecified information need.

Based on their structure, index expression can be broken down into subexpressions. Roughly speaking, index expression I is a subexpression of another IE J, if I is contained in J. This is denoted by the so called subexpression relation, represented by the symbol ≪. The expression I ≪ J means that I is a subexpression of J. Alternatively, J is a superexpression of I. Subexpressions with smallest possible difference are called direct subexpressions, denoted by the direct subexpression relation ≪d.

Example 4.2 This example illustrates the intuition behind the intended notions of subexpressions. Figure 4.2 sketches the ways in which leaves of an IE can be removed to obtain subexpressions.

From a set of index expressions, a navigational overview structure can be constructed. Such navigational overview structures are called lithoids ([Bru90]) because of their crystalline structure. Lithoids can be seen as graphs as illustrated in Figure 4.3 which shows the lithoid for the IE of Example 4.2. 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.
**Example 4.3** The index expression of example 4.2 and all its subexpressions form a lithoid, as depicted in Figure 4.3. The lines represent direct subexpressions. The solid lines denote direct subexpressions that are obtained by defoliation, i.e. removal of a leaf. The dotted lines denote direct subexpressions that are obtained by removing the lead term. Note that removing the head corresponds to a topic shift since the subexpression features a different lead term. The lead term may only be removed if it has only a single subexpression.

![Defoliation](image)

**Figure 4.3:** Example lithoid.

A number of ways in which lithoids are exploited is presented next. Navigational overview structures for collections of documents are obtained by combining lithoids for single index expressions through node sharing ([Bru93]). Lithoids that are constructed from document characterisations serve as hyperindex in 2-level hypermedia representations (see e.g., [BW92, BW90]). In this hypermedia structure, documents in the base layer are connected to the IEs of their characterisation. In this way, stratified hypermedia representations of ID systems are obtained. See Figure 4.4 for a graphical sketch of this hypermedia model. A transition from the lithoid down to the base layer is called beam down. A transitions in the opposite direction is coined beam up.

Lithoids support navigational query formulation by a process coined *Query by Navigation* (QBN) (see e.g., [BW90]). In QBN, one starts at some node (index expression) in the lithoid and then navigates through the lithoid by repeatedly selecting an adjacent link that leads to a different index expression. 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...
4.2. FUNDAMENTALS OF INDEX EXPRESSIONS

Figure 4.4: Stratified architecture.

direct subexpressions. This 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 described in Section 4.3.2, may be exploited for constructing such an algorithm for enlargements.

QBN has been implemented in a number of information systems. An IR system that supports QBN on the Internet is described in [IWW+95]. In Chapter 6, a more recent system is described. An implemented retrieval engine based on QBN for the disclosure of a slides library is described in [BBB91]. Furthermore, QBN still receives attention in ongoing research. For instance, mechanisms for navigational support during QBN are described in [Ber98, BB97, BHW94]. 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. Several search engines for the WWW, including HotBot and AltaVista, feature a QBN-like approach to query formulation. Concretely, they allow the user to refine his query by selecting related topics that are generated from a network.

Logical frameworks for QBN search paths have been developed. Preferential Models for IR, capable of reasoning with user non-monotonic preferences, were described in Section 2.2.3. It was mentioned that QBN search paths describe Preferential Models. In this sense, Preferential Models depends on properties of user preferences specified in terms of index expressions. Therefore, a clear definition of (properties of) index expressions is required.

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 intuition of these issues may be clear, sound formal ground has not been developed yet. Therefore, sections 4.3, 4.4, and 4.5 are devoted to lifting this lack.
4.3 Nesting and Defoliation

This section provides the required formal fundamentals of index expressions, leaving several notions of subexpressions to the next section. Section 4.3.1 introduces index expressions based on the nesting operator and provides a number of properties of index expressions. Section 4.3.2 elaborates the defoliation of index expressions, i.e. the removal of their leaves. Defoliation is the key notion with which subexpressions are defined in the next section. In addition, our research supports the design of algorithms for computing enlargements and refinements. In particular, the defoliation operator can be readily implemented in order to obtain an algorithm for (direct) enlargements.

This section has several aims. Next to providing a proper answer to the presumptions on which the applications of index expressions are based, it enhances insights in the process of defoliating index expressions and derived notions of subexpressions. In addition, it opens doors for new applications by easing the analysis of properties of IEs. For example, different notions of subexpressions are defined through defoliation.

4.3.1 Structural Representation of Index Expressions

We make an explicit distinction between the empty index expression and non-empty index expressions. The empty index expression is denoted by $\epsilon$, which is a special symbol. Our definition prevents the empty index expression to appear in the set of terms or connectors on which the index expressions are based. This eases the structural definition of index expressions as well as the defoliation of index expressions.

Nonempty index expressions can incorporate a number of nested expressions. These nested expressions are non-empty index expressions themselves, allowing a recursive structure. Before defining the exact nature of non-empty index expressions, we first focus on the nested expressions.

The nesting operator for index expressions which is defined below is used to construct non-empty index expressions in Definition 4.2. The nesting operator provides a notational shorthand for denoting the nested expressions of index expressions. It defines a series of nested expressions, just like the summation operator defines a series of numbers to be added, by concatenating subexpressions from a left border up to a right border.

**Definition 4.1 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 non-empty index expressions and $c_i$ be connectors ($k \leq i \leq l$). The index expression nesting operator, denoted by $\otimes$, is defined as follows:

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

Note that the order in which the nested expressions are defined by the nesting operator is relevant. The constructor operator only defines non-empty series of nested expressions. This is a deliberate property since non-empty index expressions do not contain the empty index expression. This conforms to the original definition of index expressions as given in ([Bru93, Bru90]). We will omit parentheses if they are clear from the context.

Example 4.4 Nesting Operator For \( k = 1 \) and \( l = 2 \) we have

\[
\otimes_{i=1}^{2} c_i(I_i) = \otimes_{i=1}^{1} c_i(I_1) c_2(I_2) = c_1(I_1) c_2(I_2)
\]

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 lead term, and, possibly, a number of non-empty nested index expressions.

Definition 4.2 Language of Non-empty Index Expressions Let \( T \) be a set of terms and \( C \) be a set of connectors, such that \( T \cap C = \emptyset \) and the empty index expression \( e \notin T \cup C \). The language of non-empty index expressions based on \( T \) and \( C \), denoted by \( L_{(T,C)}^+ \), 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 \( I_1, \ldots, I_l \in L_{(T,C)}^+ \), then also \( h \otimes_{i=1}^{l} c_i(I_i) \in L_{(T,C)}^+ \).

Here, the term \( h \) is called the head or lead term. If \( T \) and \( C \) are clear, we write \( L^+ \) for short.

Note that terms and connectors may appear more than once in an index expression. Furthermore, note that the left border of non-empty 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 expressions can always be renumbered. However, note that left borders which are not equal to one are used in the defoliation of index expressions (see case (4.5) of Definition 4.5).

To allow for the construction of multi-word index expressions, the null-connector, denoted by \( \circ \), may be included as a special connector. Example index expressions that involve the null-connector are information \( \circ \) system and nesting \( \circ \) operator.

Example 4.5 Nonempty Index Expressions Since the language of non-empty index expressions is a superset of the set of terms, i.e. \( T \subseteq L_{(T,C)}^+ \), all terms are non-empty index expressions. An example non-empty index expression is

\[ E_1 = \text{industry} \]
 Furthermore, using water as head, in as connector, and the term sea as non-empty nested expression, the following non-empty index expression is constructed:

\[ E_2 = h \otimes_{i=1}^1 c_i(I_i) = hc_1(I_1) = \text{water in (sea)} \]

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

\[ E_3 = \text{rural} \circ (\text{areas}) \]

Now, using \( h = \text{pollution} \) as head, connectors \( c_1 = \text{by}, \) \( c_2 = \text{of}, \) and \( c_3 = \text{in}, \) and non-empty nested expressions \( I_1 = E_1, \) \( I_2 = E_2, \) and \( I_3 = E_3, \) we obtain

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

\[ = \text{pollution by (industry) of (water in (sea)) in (rural} \circ (\text{areas})) \]

A definition of index expressions that does allow non-empty index expressions to contain empty subexpressions is given in [Ber98]. 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 \( e \) and \( e \) in \( e \) are valid (generalised) index expressions. This may be exploited for representing partial knowledge. 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 non-empty 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 non-empty index expressions united with the empty index expression.

**Definition 4.3 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.

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 each occurrence of a term is counted. The empty index expression has size 0 since it does not contain any terms. In the size of a non-empty index expression, the head as well as the sizes of the nested expressions are counted. A leaf is a non-empty index expression without nested expressions. That is, leaves coincide with terms. This is captured in the following Boolean function

\[ \text{IsLeaf}(I) \leftrightarrow I \in T \]

This function is used in the following section in the definition of defoliation.
4.3.2 Defoliation of Index Expressions

In this section, we consider the defoliation of index expressions. Defoliation, meaning the removal of leaves, is an important concept for the use of IEs 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 ID. 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. Including this in the definition of subexpressions is useful for, for instance, small IEs which otherwise generate few subexpressions. Removal of the head may occur if it has a single nested expression. In the remainder of this section, we implicitly include this case when referring to leaves. However, it may be left out if required.

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 non-empty sequence of natural numbers, the elements of which iteratively specify the subexpression in which the leaf resides. For instance, in index expression $E_4$ of example 4.5, the sequence $[2, 1]$ denotes the leaf sea since this term is obtained by first selecting the second subexpression, i.e., $I_2 = \text{water in } (\text{sea})$, 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 non-empty sequences of natural numbers can identify leaves, we call them pointing sequences.

Definition 4.4 Nonempty Sequences of Natural Numbers Let $N$ denote non-negative integers, i.e. $0, 1, 2, \ldots$. Furthermore, let $\tilde{N}$ denote the set of non-empty sequences over non-negative integers. 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 non-empty tail of the sequence. We also use variables like $\tilde{x}$ to denote a sequence.

Example 4.6 Pointing Sequence Figure 4.5 shows index expression $E_4$ equipped with pointing sequences for all its terms. For example, the sequence $[0]$ points to the head pol-
The sequence $[2, 1]$ points to the term sea, which is a leaf. Pointing sequence $[3]$ specifies term rural, which is not the head nor a leaf.

![Image of example index expression with pointing sequences for all terms.]

Pointing sequences and natural numbers do not add expressive power to the definitions in this section because index expressions, in particular the sizes of their nested expressions, can serve the same purpose. 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 for reasons of presentational simplicity without relying on additional power or properties that were not catered for by index expressions themselves.

Each leaf can be identified as the rightmost leaf in the nested expression that covers the part of the original index expression left to the upward path from that leaf to the head. This is illustrated in Figure 4.6(a), where the grey area models the indicated nested expression. So, for the sake of identifying leaves, pointing sequences are not strictly necessary. They do, however, provide 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 nested expressions of $I$ are mapped to elements of a pointing sequence. The mapping is illustrated in Figure 4.6(b). In this way, the nested expressions 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.
Defoliation Operator.

The defoliation operator $\Delta$ removes a leaf which is indicated by a given pointing sequence. The elements of the sequence denote which nested expressions to visit in order to reach the leaf. For an index expression $I$ and a non-empty 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 4.5 Defoliation of Index Expressions** The defoliation operator $\Delta$ maps an index expression and a non-empty sequence of natural numbers to an index expression:

$$\Delta : \mathcal{L} \times \tilde{N} \to \mathcal{L}$$

The definition of the defoliation operator is given by (4.1)-(4.5) below by examining the different cases that may occur.

The defoliation operator is defined inductively, 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:

$$\Delta(h, [0]) = \epsilon \quad (4.1)$$
The second non-recursive case deals with an index expression that has exactly one nested expression. If the head of this index expression is removed, by defoliating it at position 0, only the nested expression remains.

\[ \Delta(hc_1(I_1), [0]) = I_1 \]  

(4.2)

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

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

(4.3)

Example 4.7 Non-recursive Cases of Defoliation  
This example illustrates cases (4.1), (4.2), and (4.3), respectively.

Defoliating a term results in the empty index expression:

\[ \Delta(\text{industry}, [0]) = \varepsilon \]

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

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

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

\[ \Delta(\text{water in (sea), [1]} = \text{water} \]
\[ \Delta(\text{pollution of (water) by (industry), [1]} = \text{pollution by (industry)} \]
\[ \Delta(\text{pollution of (water) by (industry), [2]} = \text{pollution of (water)} \]

As another example, consider index expression \( E_4 \) from example 4.5:

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

The recursive case for defoliation is first illustrated in example 4.8. Then, the definition is followed by an example providing a number of practical illustrations of the recursive case of defoliation.
Example 4.8 Recursive Defoliation (Schematic) Figure 4.7 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 nested expression $I_x$ 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 nested expressions 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 4.7 by the left and right triangles. Subtree $I_x$, depicted by the large triangle, hosts the remainder of the defoliation. The call that effectuates this, $\Delta(I_x, [xs])$, will recursively locate 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 4.7: Schematic description of recursive defoliation.](image)

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 nested expression: $l \geq 1$. Furthermore, the defoliation must take place in an existing nested expression. 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 nested expression $I_x$ 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 nested expressions left of the selected nested expression $I_x$ ($\otimes_{i=x+1}^l c_i(I_i)$) are copied. Connector $c_x$, that connects the head to the selected nested expression, is copied as well. Then, defoliation proceeds by the recursive call $\Delta(I_x, [xs])$ which removes the leaf that is denoted relatively in nested expression $I_x$ 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 nested expressions right of $I_x$ ($\otimes_{i=x+1}^l c_i(I_i)$) are copied:
\[
\Delta(h \otimes_{i=1}^{x-1} c_i(I_i), [x,xs]) = h \otimes_{i=1}^{x-1} c_i(I_i) \ c_x(\Delta(I_x, [xs])) \otimes_{i=x+1}^{l} c_i(I_i)
\] (4.4)

left subs right subs

If \( I_x \) were to be a leaf, the recursive application of the defoliation operator could later result in leaving an empty descriptor \( \epsilon \) in the index expression, which is not allowed in valid index expressions (see Definition 4.3). The \texttt{not IsLeaf}(I_x) is thus necessary to prevent the use of \( \epsilon \) as nested expression. 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 (4.4) occur if there is only one nested expression or if defoliation is to proceed in the first or last nested expression. This is because the nesting constructor \( \otimes \) always renders a non-empty result. In the case defoliation is to proceed in the first nested expression, the part \( \otimes_{i=1}^{x-1} c_i(I_i) \) is skipped. In the case of proceeding in the last nested expression, the part \( \otimes_{i=x+1}^{l} c_i(I_i) \) is left out. In the case of a single nested expression, both parts are left out.

**Example 4.9 Applications of Recursive Defoliation**

Again, consider

\[
E_4 = \text{pollution by (industry) of (water in (sea)) in (rural o (areas))}
\]

Suppose we want to remove the leaf \texttt{sea} from \( E_4 \). This leaf is denoted by pointing sequence [2,1] (see also Fig. 4.5).

The first call,

\[
\Delta(\text{pollution by (industry) of (water in (sea)) in (rural o (areas))}, [2,1])
\]

is covered by the recursive case. The recursive call that results from applying this case, \( \Delta(I_2, [1]) \), proceeds in nested expression \( I_2 = \text{water in (sea)} \). In \( I_2 \), leaf \texttt{sea} is denoted by the pointing sequence [1]. The leftmost nested expressions, here only \( I_1 = \text{industry} \), and the rightmost, here \( I_3 = \text{rural o (areas)} \) are not altered. Thus, after this first step we have

\[
\text{pollution by (industry) of(\Delta(\text{water in (sea)}, [1])) in (rural o (areas))}
\]

The second application of the defoliation operator, \( \Delta(\text{water in (sea)}, [1]) \), was illustrated in example 4.7 that showed that the result of this is the single term \texttt{water}. The final result therefore becomes

\[
\text{pollution by (industry) of (water) in (rural o (areas))}
\]
So far, this section has described the valid cases of defoliation. All other cases of defoliation explicitly render the value undefined:

\[
\Delta(I, \vec{x}) = \text{undefined, otherwise} \quad (4.5)
\]

**Example 4.10 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 nested expression that is out of range is also invalid. Such pointing sequences go beyond the width of the index expression.

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

\[
\Delta(e, [1])
\]

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 nested expressions, the largest prefix of the pointing sequence is processed correctly. In the end, however, defoliation can no longer proceed and, for some term \( t \in \mathcal{T} \) and pointing sequence with at least two elements \([x, xs]\) results in

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

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

\[
\Delta(he1(I_1)e2(I_2), [3, xs])
\]

We say that \( \Delta(I, \vec{x}) \) 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 \( \Delta \) in the remainder of this section.

**Properties of Defoliation.**

In this section, 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 4.1 (One Term Less) For every non-empty 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:** Consider a defined defoliation $\Delta(I, \vec{x})$. Defoliation terminates in case (4,1), (4,2), or (4,3). 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 (4.5) is applied, after which the computation terminates. If the defoliation is defined, case (4.5) does not occur. In the recursive case (4.4), 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 (4.1), (4.2), or (4.3).

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.

### 4.4 Representational Formalisms

Index expressions can be represented in several ways. We shall discuss the structural representation, the grammar representation, and the broaden-based representation. Each representation has its own advantages and disadvantages. As is shown in this section, the representations describe the same language of index expressions but differ in denotational properties. The following sections are intended to help in choosing the most suitable representation for a specific task. In Section 4.6, denotational properties are exploited in describing different matching strategies for index expressions.

In section 4.3.1, the structural representation of index expressions was introduced. 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 4.3.

#### 4.4.1 Structural Representation

In section 4.3.1, we introduced the structural representation of index expressions, which is based on the nesting operator $\otimes$. The structural representation serves well if all subexpressions at a certain depth have to be addressed at the same time. In this sense, the structural
4.4. REPRESENTATIONAL FORMALISMS

representation provides a vertical decomposition of index expressions, allowing a direct and clear look on their structure.

The structural representation is exploited in cases where the order of subexpressions is to be taken into account. This occurs, for instance, in section 4.6.3, where the similarity between two index expressions involves a maximisation over all subexpressions.

4.4.2 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. Definition 4.6 (taken from [Bru93]) provides a grammar representation for index expressions.

**Definition 4.6 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):

\[
\begin{align*}
\text{Expr} & \rightarrow \epsilon | \text{NExpr} \\
\text{NExpr} & \rightarrow \text{Term} \{\text{Connector (NExpr)}\}^* \\
\text{Term} & \rightarrow \text{t}, \text{t} \in T \\
\text{Connector} & \rightarrow c, c \in C
\end{align*}
\]

In this definition, $\text{Expr}$ stands for index expression and $\text{NExpr}$ for non-empty index expression. The definition is illustrated in the next example.

**Example 4.11** As the first line of the grammar shows, index expressions are either empty or can be generated by the non-terminal $\text{NExpr}$. This non-terminal generates a non-empty index expression by glueing a lead term together with several subexpressions, as described by the second line of the grammar. This is illustrated in Figure 4.8, giving a parse tree representation of an example index expression. The generated subexpressions may contain nested $\text{NExprs}$. Note that the grammar allows for a lead term without nested subexpressions. The leaves of the parse tree denote terms and keywords, which are generated by the last two rules of the grammar.

The grammar representation is also called the abstract syntax representation. It describes the essence of the structure of index expressions. It should be noted that the abstract syntax is not intended for parsing text in order to obtain index expressions. For that, a more detailed concrete grammar is required.

The grammar representation is equivalent with the structural representation. Equivalence, here, means that both representations generate the same language.
Theorem 4.1 Grammar Representation $\equiv$ 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 $\text{Expr} \rightarrow \epsilon$ describing the empty index expression $\epsilon$ which is also in the structural representation.

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

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

The induction hypothesis is that all parse trees starting with $\text{Expr} \rightarrow \text{NExpr}$ which contain at most $n$ occurrences of the $\text{NExpr}$ 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 $\text{NExpr}$ nonterminal. This parse tree starts with $\text{Expr} \rightarrow \text{NExpr} \rightarrow \text{Term} \{\text{Connector (NExpr)}\}^l$ for some $l \geq 0$. By the induction hypothesis, all $\text{NExpr}_i$ ($1 \leq i \leq l$) are the root of a parse tree that contains at most $n$ occurrences of the $\text{NExpr}$ 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 $\text{Term}$, which is covered by the basis step of the induction, and the $\text{Connector}$ 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.

### 4.4.3 Inductive Broaden-Based Representation

The inductive representation, as described in definition 4.7, most basically describes the (de)composition of index expressions. It is directly based on the idea that index expressions can be augmented with subexpressions through connectors, as illustrated in Figure 4.9. In this sense, the inductive representation provides a horizontal decomposition of index expressions: subexpressions are added to the right.

The broaden-based representation is defined in terms of the binary broaden operator $\text{add}$. This operator broadens an index expression $I$ with a connector $c$ and an index expression $J$, and results in a larger index expression $\text{add}(I, c, J)$. The broaden operator, see Fig. 4.9, is used in the parsing mechanism described in [Bru93].

![Figure 4.9: Basic setup of inductive representation.](image)

An advantage of this representation lies in the fact that the broaden 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. However, 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 ([WBWb]).

The broaden operator can be exploited in an inductive definition of index expressions.

**Definition 4.7** Let $T$ be a non-empty set of terms and $C$ be a set of connectors such that $T \cap C = \emptyset$. Then, the language of non-empty index expressions $\mathcal{L}^+$ is defined as:

1. if $t \in T$, then $t$ is a non-empty index expression, and
2. if $I$ and $J$ are non-empty index expressions and $c \in C$ is a connector, then $\text{add}(I, c, J)$ is also a non-empty index expression, and
3. no other non-empty index expressions exist.
Note that definition 4.7 only caters for non-empty index expressions. This ensures that the empty index expression, denoted by $\epsilon$, cannot appear within a non-empty index expression. This adheres to the notion of index expressions as described in [WBW00]. The (total) language of index expressions comprises all non-empty index expressions and the empty index expression.

Example 4.12 Single word queries or document representations are modeled by terms. Example terms are conference, biology, and Holland. Composed index expressions can be constructed through the add operator. This also exploits connectors, such as in, with, and on. For instance, the composed index expression $\text{add(}\text{conference, on, biology}\text{)}$ might represent information on a conference on biology. As a more complex example, consider

$$\text{add(}\text{add(}\text{conference, on, biology}\text{), in, Holland}\text{)}$$

that may denote information on a conference on biology held in Holland. The semantics of index expressions depends on their structure. That is, differences in nested subexpressions may cause differences in semantics. As an example of this, compare the last index expression with the slightly different

$$\text{add(}\text{conference, on, add(biology, in, Holland)}\text{)}$$

The last one more likely describes a conference about biology as far as it is practiced in Holland.

As a more complex example, consider

$$\text{add(}\text{add(}\text{add(pollution, by, industry), of, add(water, and, air)}, \text{in, add(rural, o, areas)}\text{)}\text{)}$$

which is the broaden-based representation of IE $E_4$ from example 4.5.

Similar to the structural definition, the generated language of non-empty index expressions is augmented with the empty one to obtain the complete language of index expressions.

Definition 4.8 Broaden-based Language of Index Expressions

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

The structural and the inductive representation can be transformed into one another as sketched by the scheme below.

$$h \otimes_{i=1}^{l} c_i(I_i) \leftrightarrow \text{add(}\ldots\text{add(}\text{add(}h, c_1, J_1\text{), }c_2, J_2\text{)}\ldots, c_l, J_l\text{)}$$

This scheme represents the same principle of transforming a tree into a binary tree, or vice versa. It will be exploited in proving the equivalence between the broaden-based and the structural representations.
4.4. REPRESENTATIONAL FORMALISMS

Theorem 4.2 Broaden-based $\equiv$ Structural Representation

Proof:

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 4.1. 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 4.8.

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 $I = h \otimes_{i=1}^{k} c_i(I_i)$ 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:

$$\text{add}(\ldots \text{add}(\text{add}(h, c_1, J_1), c_2, J_2) \ldots, c_l, J_l)$$

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

An advantage of the inductive representation is that it allows several auxiliary functions to be readily designed. An advantageous feature of the inductive representation as data structure is the stable number of arguments of the add operator.

Three auxiliary functions on index expressions are introduced in figure 4.10, yielding the terms of an index expression, its connectors, and its head, respectively. These functions are used in defining similarity measures in section 4.6. The definitions illustrate the elementary decomposition of the inductive representation of index expressions.
4.5 Subexpressions

Subexpressions of index expressions play an important role in many applications of index expressions in IR. For instance, the construction of lithoids 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.

In the following subsections, four 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.

4.5.1 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 4.9 Direct Subexpression Relation**

$$I \prec_d J \iff \exists \bar{x} \in \bar{N} : \Delta(J, \bar{x}) = I$$

The following lemma holds for the direct subexpression relation:

**Lemma 4.3 ($\prec_d$ is asymmetric)**
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 [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, i.e. direct refinements and enlargements, 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 [BL97] and [WHHW96], constructs Preferential Models out of search paths.

In addition, the direct subexpression relation serves as the basis for defining the edges of more complicated navigational structures such as association indices ([WBW98a, WBW98b]). This topic is further elaborated in Section 6.7. 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.

### 4.5.2 Strict Subexpressions

The strict subexpression relation for index expressions \( \preceq \subseteq L^2 \) is defined as the strict transitive closure of the direct subexpression relation. The strict transitive closure of a relation \( R \) is denoted by \( R^+ \).

**Definition 4.10 Strict Subexpression Relation**

\[
\preceq = \preceq^+_d
\]

The following lemma is now clear:

**Lemma 4.4 (\( \preceq \) is asymmetric and transitive)**

The strict subexpression relation can be exploited for personalising search paths in QBN ([Ber98]). 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.
4.5.3 General Subexpressions

The third relation on index expressions defined in this section is the (general) subexpression relation $\subseteq \subseteq \mathcal{L}^2$ 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. For convenience, the subexpression relation is represented by a set of pairs.

**Definition 4.11 Subexpression Relation**

$$\subseteq = \subset \cup \{(I, I) | I \in \mathcal{L}\}$$

The following lemma holds for general subexpressions:

**Lemma 4.5 ($\subseteq$ is reflexive, antisymmetric, and transitive)**

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 ([Won96]). 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. In the next section, we describe a non-contiguous notion of subexpressions.

4.5.4 Embedding

Embedding or containment plays a prominent role in IR. For example, an often applied strategy to query-document matching is: if the query is contained or embedded in a document, then the document is deemed relevant to the query.

Different notions for embedding exist. The subexpression relation for index expressions, for example, defines a connected variant of embedding: a subexpression is a connected part of
its superexpressions. For instance, *surfing in Holland* is a subexpression of *surfing in Holland in November* but not a subexpression of *surfing in north of Holland*.

In the abovementioned case, *north* modifies the last term *Holland*. In our view, *surfing in Holland* is therefore embedded in *surfing in north of Holland*. To cater for these cases, we exploit a slightly more liberal version of embedment.

Connectedness can, in the context of subexpressions, be described by direct ancestorship, or, parenthood. This means that for all terms in an index expression \( I \), their parent must also be their parent in index expressions in which \( I \) is embedded. Instead of direct ancestorship we use the notion of (general) ancestorship. This means that index expression \( I \) is embedded in \( J \) iff for all terms in \( I \) their ancestors in \( I \) are also ancestors in \( J \).

These considerations are reflected in our notion of embedding of index expressions which is formalised by an embedment \( \ll \) relation as follows.

**Definition 4.12** Embedding of index expressions is captured in the binary relation \( \ll \), where \( I \ll J \) means that \( I \) is embedded in \( J \), which is defined as:

- **(Same)** \( t \ll t \)
- **(Sub)** \( I \ll \text{add}(J, c, K) \) if \( I \ll J \) or \( I \ll K \)
- **(Split)** \( \text{add}(I, c, J) \ll \text{add}(K, d, L) \) if \( c = d \) and \( I \ll K \) and \( J \ll L \)
- **(Stop)** no other cases apply

Terms are embedded in themselves, as described by case **Same** of definition 4.12. Case **Sub** states that an index expression is embedded in a composed one \( \text{add}(J, c, K) \), if it is embedded in either subexpression \( J \) or \( K \). The third case **Split** shows that a composed index expression \( \text{add}(I, c, J) \) is embedded in another composed index expression \( \text{add}(K, d, L) \) if their connectors are equal, the leftmost subexpression \( I \) is embedded in the other leftmost subexpression \( K \), and a similar argument holds for the rightmost subexpressions \( J \) and \( L \).

### 4.6 Matching

Comparing descriptors is a core function of information retrieval systems. For instance, descriptors in the characterisations of documents are compared with the query, computing their similarity, to establish the set of rendered documents. Furthermore, clustering, routing, and filtering documents also hinge on matching functions.

#### 4.6.1 Introduction

There are different ways of comparing descriptors. The resemblance between descriptors can, for instance, be viewed as logical derivations (see e.g. Section 2.2.3). In that case, derivation rules describe the way in which descriptors resemble each other. Another qualitative comparison can be based on structural properties of descriptors. For IEs, this was captured in several notions of subexpressions (Section 4.5).
Similarity functions can be viewed as operational elaborations of qualitative aspects of resemblance. In [Bru93], for example, a distance measure for IEs is derived from the number of inference steps required to transform an IE into another. The approach taken in this section also derives similarity functions from qualitative aspects of IEs.

The similarity functions are devised with three additional constraints in mind. First, the similarity measures correspond to different notions of subexpressions. The reason for this is that refinements in hyperindices directly stem from a particular view on the decomposition of index expressions. For instance, the order of subexpressions may be deemed relevant in computing refinements. In order to render documents consistent with the type of refinement selected by the user, the similarity measures should be in accordance with the underlying notion of subexpressions. We examine embedding, the order of subexpressions, and headedness. At the end of this section, we experimentally evaluate how the devised similarity measures perform in the context of subexpressions. That is, we practically examine their quality in discriminating different types of subexpressions.

Second, we aim at general-purpose matching functions that can be exploited in multi-topic domains. This means that the similarity functions are to be domain independent. This implies that the use of domain specific knowledge bases is not adequately general for our purposes.

Third, we focus on efficient similarity measures. This allows them to be used in real-time applications, such as information systems for the WWW. An example of this, the INN system, is provided in Chapter 6. As a consequence, we focus on the basic refinement structure of phrases, as captured in index expressions. In other words, we concentrate on the hierarchical decomposition of index expressions.

This section has the following structure. In section 4.6.2, related work on matching index expressions is described. Section 4.6.3 elaborates on important criteria in matching index expressions. In section 4.6.4, the actual measures are defined and analysed. Finally, section 4.6.5 provides an experimental evaluation of the devised measures.

### 4.6.2 Related Work

Matching index expressions can be done with, for instance, belief networks (see [BJ94, BG94]). The approach described does not incorporate multiple notions of subexpressions.

In addition, [Wou97, Ber98] describe a similarity function for index expressions based on their twigs, elementary combinations of terms. We build on their findings by providing an inductive definition of twigs, extending them with a depth factor, including twigs in set-based similarity measures, and comparing these with other matching functions.

In [KM93], a language for querying structured text based on tree inclusion is described. Their approach, which exploits inclusion patterns to ensure preservation of binary properties between nodes, takes both structure and content into account. Example inclusion patterns are $L$ for labels, $A$ for ancestorship, and $O$ for (left-to-right) ordered tree inclusion.
4.6. MATCHING

Our skeleton-content approach resembles their \{LAO\}-embedding. That is, ancestorship and ordering are preserved and labels are taken into account. However, our approach does not require labels to be equal but supports approximate matching by offering abstract similarity functions for terms and connectors. In the introduction of the mentioned article the authors indicate that, indeed, such “standard IR techniques should be added to the language” ([KM93]). Our similarity measures for index expressions capture both aspects by approximation: they are sensitive to both structure and, by using abstract term comparison functions, content.

Our skeleton-content approach thus ‘preserves’ labels by taking into account their similarity. Essentially, it searches for the best \{L’AO\}-embedding and delivers the degree of embedding. A similar line of reasoning shows that our full product approach computes the degree of \{LA\}-embedding. Since in the full product measure the order of subexpressions is considered irrelevant, the corresponding inclusion pattern \(O\) is not satisfied.

In [ATKW98], an approach to compare head-modifier structures is described. Lexico-semantic relations are used in computing the similarity between their binary terms. Our set based similarity measures for twigs do not explicitly make use of a semantical network. However, this may be incorporated in our abstractly modeled comparison functions for terms and connectors. In addition, twigs are tertiary structures, also including information on the type of connection between head and modifier. Furthermore, we focus on similarities between \emph{sets of} twigs, rather than on their individual similarities.

4.6.3 Criteria for Matching

The index expression

\[ hc_1(I_1) \ldots c_k(I_k) \]

can be seen as a description of a concept named \(h\) being further refined by relations called \(c_1 \ldots c_k\) and concepts \(I_1 \ldots I_k\), respectively.

As a consequence, comparison of both concepts names (contents) as well as the refining mechanism (structure) will be important issues when matching index expressions. This section presents several topics concerning contents and structure that refine the abovementioned view on the semantics of index expressions.

Contents

The contents of index expressions is given by their terms and connectors.

Terms In order to match index expressions, their terms should be compared. Therefore, we assume a similarity function between terms, denoted \(\text{sim}_T : T \times T \to [0..1]\). The
expression \( \text{sim}_T(t, t') \) denotes the similarity between the concepts referred to by terms \( t \) and \( t' \).

The similarity between terms can be obtained in several ways. For instance, it can be computed by string comparison algorithms such as \( n \)-grams (see e.g. [Bru93]), substrings of length \( n \). Furthermore, additional lexico-semantic knowledge can be used taking, for instance, hypernyms and synonyms into account. In IR, stemming (see e.g. [Pai94]) is often performed to reduce morphological variance. We abstract from the particular techniques used in computing the similarity and concentrate on using this in computing similarity between index expressions.

Connectors As is the case for terms, we also assume a similarity function \( \text{sim}_C : C \times C \rightarrow [0..1] \) between connectors. This similarity expresses the strength of the relation between connectors.

The similarity function for connectors can take several aspects into account. For instance, it can be based on the types of connectors as identified by Farradane (see [Far80a] and [Far80b]). In this approach, called relational indexing, connectors model the relationships between terms. Connectors that model the same relationship could be given a high similarity value. Furthermore, the priority of connectors within index expressions can be exploited. In addition, occurrence-frequencies of connectors could be used. Again, we abstract from the different approaches and focus on exploiting similarity between connectors for matching index expressions.

Structure

The structure of index expressions partly determines their semantics and should, therefore, be taken into account in matching. Embedded subexpressions, as described in section 4.5.4, describe a non-contiguous structural agreement between index expressions. Next to embedding, two additional issues concerning structure are considered in this section: the order of subexpressions and headedness.

Order of Subexpressions An important question considering the semantics of index expressions is whether the order of subexpressions is relevant. Consider for example the following index expressions.

\[
\text{cycling with (friends) in (mountains)} \\
\text{cycling in (mountains) with (friends)}
\]

One may argue that their meaning is equivalent. In other situations, however, there may be cases in which the order of subexpressions is relevant, i.e. causes a different meaning. For instance, if the sequential order of paragraphs in a text is represented in an index expression. The following index expressions may be argued to have different meanings.
enjoying o (drinking) after (skiing)
enjoying after (skiing) o (drinking)

The notion of order of subexpressions is formalised by the relation $\text{EqOrder}$ as follows.

**Definition 4.13** We call two index expressions $I = h \otimes_{i=1}^{k} c_i(I_i)$ and $J = h' \otimes_{j=1}^{l} d_j(J_j)$ equal modulo order, denoted by $\text{EqOrder}(I, J)$, iff

1. $h = h'$, and
2. there exists a projection $\pi$ of $[1..k]$ to $[1..l]$ such that for all $1 \leq i \leq k$ it holds that $c_i = d_{\pi(i)}$ and $\text{EqOrder}(I_i, J_{\pi(i)})$.

Two index expressions are equal modulo order iff their heads are equal and (recursively) the subexpressions of the first are equal modulo order to one of the subexpressions of the second. The second criterion holds, for instance, for index expressions of which the subexpressions (recursively) are a permutation of the other’s. In addition, it holds for index expressions which contain several copies of the same subexpressions. This is, for instance, illustrated by the index expressions retrieval of information and retrieval of (information) of (information). Such constructs can for instance be exploited to stress the importance of a subexpression. Furthermore, equality modulo order is reflexive. Later in this chapter, we describe a matching strategy that computes maximal similarity for index expressions that are equal modulo order.

**Headedness** The head of an index expression is considered to be its most important part. We will address this notion, implemented by head-modifier pairs in ([Str95, Kos99]), as headedness. Normalisation techniques may be exploited to transform pre-modifiers to post-modifiers in order to better support headedness ([Ber98]).

The subexpressions modify the main concept stated in the head. Consequently, the lower terms occur, i.e. deeper with respect to nested expressions, the less important they are. This can be taken into account in matching by exploiting the depth of terms. Multiplying by a factor that is inversely proportional to the depth gives the desired result. We will indicate how each similarity function described below can be equipped with such a depth factor.

**4.6.4 Similarity Measures for Index Expressions**

In this section, several similarity measures for index expressions are designed. They vary from simple measures that only consider the content of index expressions (section 4.6.4) to more comprehensive measures that consider both content and structure (section 4.6.4). For reasons of simplicity, cases for the empty index expressions are not always provided.
Measuring the similarity between index expressions by sets of terms and connectors considers only their contents, not their structure. Several standard measures for set similarity can be adopted to provide similarity measures for (the contents of) index expressions. Example set measures are the Dice, Cosine, and Jaccard measures (see e.g. [Rij79]). These measures do not use similarity functions for terms and connectors as described in sections 4.6.3 and 4.6.3, respectively. Instead, they use set primitives as union, disjunction, and cardinality which use equality of elements (terms and connectors). The Dice measure, which normalizes the intersection $A \cap B$ with the sum of constituents, the Cosine measure, relating the overlap of both sets to their geometric average, and the Jaccard measure, expressing the degree of overlap between two sets as the proportion of the overlap from the whole, are defined as

$$
\text{Dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \quad \text{Cos}(A, B) = \frac{|A \cap B|}{\sqrt{|A| \times |B|}} \quad \text{Jacc}(A, B) = \frac{|A \cap B|}{|A \cup B|}
$$

in case the denominator does not equal zero. In case it does, the measures return similarity value zero.

By considering the (sets of) terms and connectors of index expressions, the set based measures can be exploited for index expressions. As an example, consider the similarity measure for index expressions based on the Dice coefficient as shown in Figure 4.11.

$$
\text{Dice}_\alpha(I, J) = \alpha \times \text{Dice}(\text{Terms}(I), \text{Terms}(J)) + (1 - \alpha) \times \text{Dice}(\text{Conns}(I), \text{Conns}(J))
$$

Figure 4.11: Dice’s similarity measure for index expressions.

The factor $\alpha \in [0..1]$ determines the relative influence of terms and connectors on the similarity value. For $\alpha = 1$, only terms are considered and for $\alpha = 0$ only connectors are taken into account. Note that a depth factor cannot be taken into account directly since no information about structure is available in sets of terms.

We say a measure is maximal for certain index expressions if maximal similarity is returned for the descriptors. The Dice measure is not maximal for embedded index expressions nor for subexpressions. Consider, for example, index expression surfing in Holland and its subexpression Holland. Since the sets of terms and connectors are different, the Dice measure is not maximal for these index expressions.

Contents and Structure

This section provides three similarity measures that take both content and structure of index expressions into account. The first measure, coined *full product*, adheres to the idea
that the order of subexpressions is irrelevant for the meaning of index expressions. On the contrary, the second measure, called embedded content, deems the order relevant. In addition, embedded content is based on non contiguous embedding. Finally, a similarity measure based on twigs decomposes index expressions into elementary connections called twigs.

**Full Product** The full product similarity measure computes the degree to which an index expression is equal modulo order to another one. This means that the order of subexpressions is considered irrelevant. Since the structural representation appears most appropriate in this case, the full product measure, denoted by FP, is specified by it as depicted in Figure 4.12.

\[
\begin{align*}
\text{(Terms)} & \quad \text{FP}(t, t') = \text{sim}_T(t, t') \\
\text{(Top)} & \quad \text{FP}(t, h \otimes \prod_{i=1}^{k} c_i(I_i)) = \text{sim}_T(t, h) \\
\text{(Tall)} & \quad \text{FP}(h \otimes \prod_{i=1}^{k} c_i(I_i), h' \otimes \prod_{j=1}^{l} d_j(J_j)) = \\
& \quad \text{sim}_T(h, h') \times \max_{1 \leq i \leq k, 1 \leq j \leq l} \text{sim}_C(c_i, d_j) \times \text{FP}(I_i, J_j) \\
\text{(Toll)} & \quad \text{FP}(h \otimes \prod_{i=1}^{k} c_i(I_i), t) = \frac{\text{sim}_T(h, t)}{\text{Terms}(h \otimes \prod_{i=1}^{k} c_i(I_i))}
\end{align*}
\]

Figure 4.12: Full product algorithm.

The full product algorithm, as shown in Figure 4.12, consists of four cases. They can be readily translated to a functional algorithm by using case analysis through pattern matching. The first case Terms computes the similarity between single terms. In case Top, the embedding of a single term \( t \) in a composed index expression \( h \otimes \prod_{i=1}^{k} c_i(I_i) \) is computed by comparing \( t \) with head \( h \) since both occur at the same depth.

The third case Tall computes the similarity between two composed index expressions. Correspondingly to definition 4.13, this is the product of the similarity between the heads and the maximum similarity values of the subexpressions of \( I \) with some subexpression of \( J \).

The final case Toll gives a penalty for the fact that a composed index expression cannot be equal modulo order with a term. The returned similarity value is smaller if the mismatch in size (number of terms) is larger.

Inspection of the cases Terms and Tall of definition 4.12 shows that the full product measure is maximal for equal arguments. That is, the full product measure adheres to the elementary property of \( \text{FP}(I, I) = 1 \).

Note that the full product measure computes a similarity value layer by layer. That is, only terms and connectors that occur at the same depth are compared. The full product similarity measure is maximal for index expressions that are equal modulo order, as shown in the following theorem.
Theorem 4.3  \( \text{EqOrder}(I, J) \Rightarrow \text{FP}(I, J) = 1. \)

Proof:
The proof is done by structural induction to index expressions. We only investigate the (interesting) case of composed index expressions. Suppose \( I = h \otimes_{i=1}^{k} c_i(I_i) \) and \( J = h' \otimes_{j=1}^{l} d_j(J_j) \) are equal modulo the order of their subexpressions. This means that for each \( h c_i(I_i) \) the maximum value \( \max_{1 \leq j \leq l} \text{FP}(h c_i(I_i), h' d_j(J_j)) = 1 \) since

1. \( h = h' \), meaning \( \text{sim}_T(h, h') = 1 \), and
2. for each \( 1 \leq i \leq k \) there exists a \( 1 \leq j \leq l \) such that \( d_j = c_i \) and \( \text{EqOrder}(I_i, J_j) \)
   meaning that \( \text{sim}_C(c_i, d_j) = 1 \) and, by the induction hypothesis, \( \text{FP}(I_i, J_j) = 1. \)

This means that, for the \( j \) of case (2), \( \text{sim}_T(h, h') \times \text{sim}_C(c_i, d_j) \times \text{FP}(I_i, J_j) = 1. \) The total
   similarity measure then comes down to \( \frac{1}{k} \sum_{i=1}^{k} 1 = 1. \)

As a direct consequence of theorem 4.3, we conclude that full product computes maximal similarity for index expressions of which the subexpressions (recursively) are a permutation of the other’s or occur multiple times.

The full product measure is not maximal for embedded index expressions. Consider, for example, index expressions surfing in Holland and surfing in sunny \( o \) Holland. Since the full product measure computes similarity values layer by layer and the keyword Holland is located at different depths, the given index expressions are not maximally similar. In addition, the full product measure is not maximal for subexpressions. Illustrating this, consider Holland and surfing in Holland. Since the heads of both index expressions are unequal, the full product measure is not maximal.

\[
\text{FP}_d(h \otimes_{i=1}^{k} c_i(I_i), h' \otimes_{j=1}^{l} d_j(J_j)) = \text{sim}_d, T(h, h') \times \frac{1}{k} \sum_{i=1}^{k} \max_{1 \leq j \leq l} \text{sim}_d, C(c_i, d_j) \times \text{FP}_d+1(I_i, J_j)
\]

(4.6)

The depth factor \( d \) can be incorporated in the full product algorithm (Figure 4.12) by replacing case Tall by equation (4.6). Note that it is only shown how the depth factor is to be correctly computed in this case. Multiplying each case by a factor inversely proportional to this depth factor takes headedness into account.

Embedded Content  The embedded content measure (see Figure 4.13) computes the best way in which an index expression can be embedded (as defined in definition 4.12) in another one. This means that the order of subexpressions is considered relevant. This cases the use of the inductive representation of index expressions and thus enables an elementary decomposition of index expressions.

The cases considered by the embedded content measure are the same as for the full product measure. The case Terms of Figure 4.13 is exactly the same as for full product and calls
the similarity function for terms $\text{sim}_T$. The second case \textbf{Top} states that the degree to which a term $t$ is embedded in a composed index expression $\text{add}(I, c, J)$ is equal to the maximal similarity to one of the subexpressions $I$ and $J$. Case \textbf{Tall} computes the strength of embedding of a composed index expression $\text{add}(I, c, J)$ in another one $\text{add}(K, d, L)$. It computes the maximum similarity of the following three cases: (1) $\text{add}(I, c, J)$ is completely embedded in the leftmost subexpression $K$, (2) it is completely embedded in the rightmost subexpression $L$, and (3) subexpression $I$ is embedded in $K$, subexpression $J$ is embedded in $L$, and connectors $c$ and $d$ are similar. As before, case \textbf{Toll} gives a penalty for mismatch since composed index expressions can never be embedded in terms.

The embedded content measure is maximal for embedded index expressions, as stated more formally in the next theorem.

\textbf{Theorem 4.4} $I \preccurlyeq J \Rightarrow \text{EC}(I, J) = 1$.

\textbf{Proof:}
The theorem is proven by induction on the structure of index expressions. Suppose $I \preccurlyeq J$.

\textbf{Basis.} Suppose $I$ is a term $t$. Only the first two cases of definition 4.12 of embedded content, \textbf{Same} and \textbf{Sub}, are to be examined since the third case \textbf{Split} cannot apply ($I$ cannot be composed).

\textbf{Same:} Suppose this case of definition 4.12 applies. Then, $J = I = t$ and consequently $\text{EC}(I, J) = \text{sim}_T(I, J) = 1$.

\textbf{Sub:} Suppose this case applies which means that $J$ is composed, say $\text{add}(K, c, L)$ and thus that $t \preccurlyeq K$ or $t \preccurlyeq L$. Combining this with the induction hypothesis implies that $\text{EC}(t, K) = 1$ or $\text{EC}(t, L) = 1$. This means that, by case \textbf{Top} of the embedded content algorithm, $\text{EC}(I, J) = 1$.

\textbf{Induction step.} Suppose $I$ is a composed index expression, say $\text{add}(K, c, L)$. Furthermore, without loss of generality, let $J = \text{add}(M, d, N)$. Now, only cases \textbf{Sub} and \textbf{Split} of definition 4.12 need to be examined. Both result in $\text{EC}(I, J) = 1$:}

\begin{align*}
\text{(Terms)} & \quad \text{EC}(t, t') = \text{sim}_T(t, t') \\
\text{(Top)} & \quad \text{EC}(t, \text{add}(I, c, J)) = \max\{\text{EC}(t, I), \text{EC}(t, J)\} \\
\text{(Tall)} & \quad \text{EC}(\text{add}(I, c, J), \text{add}(K, d, L)) = \\
& \quad \quad \max\{\text{EC}(\text{add}(I, c, J), K), \text{EC}(\text{add}(I, c, J), L), \\
& \quad \quad \quad \text{EC}(I, K) \times \text{sim}_C(c, d) \times \text{EC}(J, L)\} \\
\text{(Toll)} & \quad \text{EC}(\text{add}(I, c, J), t) = \frac{\text{sim}_T(\text{Head}(I), t)}{|\text{Terms}(\text{add}(I, c, J))|}
\end{align*}
Sub: Here, we have (1) $I \ll M$ implying $\text{EC}(I, M) = 1$ or (2) $I \ll N$ which means that $\text{EC}(I, N) = 1$. Since case Tall of the algorithm is applied, which computes the maximum of these cases, both options cause $\text{EC}(I, J) = 1$.

Split: In this case, we have that $c = d$, $K \ll M$, and $L \ll N$. By the induction hypothesis, this implies that $\text{EC}(K, M) = 1$ and $\text{EC}(L, N) = 1$. Together with $\text{sim}_C(c, d) = 1$ this causes case Tall of the embedded content algorithm to compute $\text{EC}(I, J) = 1$.

We have shown that in all possible cases in which $I$ is embedded in $J$ the embedded content measure computes $\text{EC}(I, J) = 1$.

The embedded content measure is also maximal for subexpressions. This stems from the observation that every index expression that is a subexpression of another is also embedded in it. This, in turn, follows from the observation that in the subexpression relation, direct ancestorship is preserved, which is stricter than the general ancestorship which is preserved by the embedment relation. In addition, the embedded content measure is maximal for equal arguments, since every index expression is embedded in itself, i.e. embedment is reflexive.

The embedded content measure is not maximal for index expressions that are equal modulo order. Consider, for example, the following pair of index expressions that are equal modulo the order of their subexpressions: $I = \text{conference on (biology) in (Holland)}$ and $J = \text{conference in (Holland) on (biology)}$. Since in the embedded content measure the order of subexpressions is relevant, index expressions $I$ and $J$ are not maximally similar.

$$\text{EC}_d(\text{add}(I, c, J), \text{add}(K, d, L)) = \text{EC}_d(I, K) \times \text{sim}_{d+1}(c, d) \times \text{EC}_{d+1}(J, L) \quad (4,7)$$

The depth factor $d$ can be incorporated in the inductive representation by the scheme of equation (4.7). Note that the rightmost subexpressions $J$ and $L$ reside one layer deeper than the head.

Twig-based Similarity The twigs (see [Ber98]) of an index expression are its subexpressions that consist of exactly two terms and one connector. Twigs are the elementary connections in the concept graph which is formed by an index expression. Twigs have essentially the same structure as the head-modifier pairs described in [Str95], except that they also contain a connector. Twigs enable us to form a global picture about similarity while focusing on the elementary refinements.

To denote subexpressions of size two, we use the subexpression relation as described in section 4.5.4.

$$\text{twigs}(I) = \{tct' | tcl' \ll I\}$$
Twigs can be defined constructively in terms of the inductive representation of index expressions. This shows that the twigs of an index expression can be produced by a straightforward syntactic process. Below, twigs are accompanied by their depth factor, modeled by a positive integer. Since the head of an index expression is assigned depth one, the expression $\text{twigs}(I,1)$, according to the definition below, computes the twigs of $I$ with their correct depths.

$$
\begin{align*}
\text{twigs}(t,d) &= \emptyset \\
\text{twigs}(\text{add}(I,c,J),d) &= \{\text{add}(\text{Head}(I),c,\text{Head}(J)),d\} \cup \text{twigs}(I,d) \cup \text{twigs}(J,d+1)
\end{align*}
$$

Single terms do not contain subexpressions of size two and thus have no twigs. Composed index expressions $\text{add}(I,c,J)$ lead to at least one twig: $\text{Head}(I)c\text{Head}(J)$. Additional twigs may result from the subexpressions, as modeled by the recursive calls.

As observed in [Ber98], twigs conserve most of the structure of index expressions. In fact, if all terms are different the complete structure can be correctly reconstructed without any additional information. For twigs with depth factor, this can be relieved further to all terms at the same depth should be different.

Twigs of index expressions seem to resemble tri-grams for strings. Considerations similar to tri-grams are therefore expected to hold for twigs. For instance, an advantage of the use of twigs over equality-matching is their robustness for ‘spelling variations’. For example, if two index expressions are equal modulo order, then their twigs are the same, as stated in the following lemma.

**Lemma 4.6** $\text{EqOrder}(I,J) \Rightarrow \text{twigs}(I,1) = \text{twigs}(J,1)$.

Set-based approaches, described for terms in section 4.6.4, can also be equipped with twigs. Instead of calling the set based approaches with sets of terms or connectors as arguments, the (sets of) twigs are used. This yields similarity measures using twigs. As an example, the Dice measure for twigs is defined as

$$
\text{Dice-twigs}(I,J) = \text{Dice}(\text{twigs}(I,1),\text{twigs}(J,1))
$$

The Dice measure for twigs compares sets of twigs. As such, it compares elements that contain (some of) the structure of IEs. The Dice measure for twigs is maximal for index expressions that are equal modulo order and for identical index expressions, since their twigs are equal.
4.6.5 Experimental Evaluation

Subexpressions are the basic notion on which the hyperindex is generated. The INN system (see Chapter 6) thus heavily uses subexpressions. We therefore evaluate how the different similarity measures behave in the context of subexpressions. A number of experiments, described in this section, give insight in this matter.

The experiments exploit a non-trivial index expression (see Figure 4.14(a)) consisting of 11 terms nested in subexpressions of different forms. The similarity between this index expression and all its subexpressions was computed, according to the different similarity measures. Both contiguous and non-contiguous (e.g. workshop during November) subexpressions were generated from the original. In total, the experiment featured 439 such subexpressions. Similar tests may be performed with other notions of subexpressions. For instance, the order of subexpressions may be varied.

Figure 4.14(b) gives an overview of the different similarity measures. The graphs provided in this figure serve as reference for the next sections. Not all set-based measures are provided; the Dice measure (for terms and twigs) is given as representative. For several reasons, the subexpressions on the x-axis are sorted according to their similarities such that non-increasing graphs are obtained. Sorting similarities non-increasingly and therefore grouping together subexpressions that are equally similar to the original index expression in intervals, clearly shows the results of the similarity measures. Without the sorting, different measures in the same graph would be hard to distinguish. Furthermore, since subexpressions within the same interval have the same similarity value, intervals rather than the positions on the x-axis should serve as indication of the ordering. For instance, the number of intervals gives an indication of the granularity of the measure. We therefore compare similarity measures by examining differences between their intervals. The order of subexpressions on the x-axis may be different for different similarity measures since we sort non-increasingly per measure. The different orders per measure are not explicitly visualised in the graphs.
Therefore, our examination used detailed information, as kept in the test results, next to the graphs. The differences between the measures are examined in the rest of this section. In our experiments, we examine differences between similarity measures by imposing the order of one similarity measure (the so called dominant measure) onto other similarity measures (the subordinate measures). That is, the order of subexpressions on the x-axis is set to conform to the non-increasing order of the dominant measure. According to this ordering, the subordinate measures are plotted. In this way, the differences in ordering are visualised.

Figure 4.15: Generic sketch of results.

Figure 4.15 gives a generic sketch of the graphs that resulted from the experiments. In this figure, the solid line depicts the similarity values of the dominant measure. In later figures, where actual similarity measures are used, the line of the dominant measure equals its similarity graph of Figure 4.14(b): the subexpressions on the x-axis are sorted such that a non-increasing line for the dominant measure is obtained. According to this dominant ordering, the subordinate measures are plotted. Note that the example dominant measure has three horizontal plateaus, i.e., intervals of subexpressions that all receive the same similarity value, or, which are not discriminated by the dominant measure. For subexpressions out of the same plateau, the differences in positions on the x-axis are thus irrelevant to the dominant measure. However, a subordinate measure may assign different similarity values to subexpressions out of the same dominant plateau. Therefore, we re-sort subexpressions per dominant plateau according to subordinate measures. In this way, we obtain non-increasing graphs for the subordinate measures per dominant plateau. Note that the number of subordinate plateaus may differ per dominant plateau. However, this number at most equals the number of plateaus in the original graph of the subordinate measure of Figure 4.14(b). As a result of sorting non-increasingly, all peaks in subordinate graphs occur at the beginning of dominant plateaus. The form of actual graphs provided later conforms to the generic sketch of Figure 4.15 but has much finer granularity because of more and smaller plateaus.
We first compare the measures that use both content and structure: the full product, embedded content, and set based measure for twigs. In sections 4.6.5 and 4.6.5, respectively, the full product and embedded content are used as primary measure. Set-based measures for twigs are compared in section 4.6.5.

**Full Product**

The results of imposing the full product ordering onto the embedded content and Dice’s measure for twigs are shown in Figure 4.16. The three measures define different orderings, as can be concluded from the many peaks (embedded content) and drops (Dice-twig) with respect to their original lines in Figure 4.14(b).

The ordering imposed by full product is related to the removal of leaves starting at maximal depth, or, a layer by layer (bottom up) defoliation. Thus, the full product measure scores high if the heads are (recursively) equal. In other words, a subexpression that equals a topmost (layer by layer) part of the original gets a high score. Full product thus demands the main concepts to be equal and is less sensitive for modification of the most specific (maximal depth) modifiers. The gradually decreasing graph of the full product measure combined with its high granularity implies that this measure delicately distinguishes most subexpressions.

The rightmost interval of full product is a large zero-plateau. In this large interval, the subordinate measures resemble their original graphs of Figure 4.14(b). Inspection of the test results showed that the subexpressions in this plateau, compared to the original, either have a different head or contain a single subexpression that has a different head. For comparing index expressions that represent different main concepts, the full product measure is thus inadequate.

![Figure 4.16: Full product order imposed.](image)
4.6. MATCHING

Dice measure for twigs. By examining which subexpressions cause these drops and peaks, the differences between the measures are studied.

Basically, two types of subexpressions cause the peaks for the embedded content measure. Of both types, a stereotypical example is investigated. Consider, for instance, the peak in Figure 4.16 that occurs at position 43, having similarity value 0.5. The subexpression causing this peak consists of 10 terms, missing only the depth two leaf Amsterdam. However, full product favours subexpressions which are obtained by removing leaves at larger depth. The many possibilities to construct subexpressions in this way makes that full product has many higher scoring subexpressions. As concluded from analysing the test results, embedded content would have placed it in the interval at positions 2-5 of Figure 4.14(b). The removal of the highest leaf (i.e. Amsterdam) is, to embedded content, similar to removing a leaf at largest depth, such as disk. Similar subexpressions cause the peaks at positions 17, 80, 101, and 171. Thus, when the influence of concept modification is desired to gradually decrease with depth, full product is a better measure than embedded content.

Another type of subexpression causes the peak at position 277, which is visually somewhat obscured by a simultaneous drop in the graph of the Dice-twig measure. This peak of similarity value 0.25 is surrounded by values of 0.09. It is caused by an index expression that does not have workshop as head, but has a large embedded subexpression consisting of 7 terms. Since the main concepts are different, full product assigns a low score. However, embedded content assigns a higher score and resorts it to the beginning of the value zero interval. Embedded content, by its own order, would have placed it in the interval at positions 2-5. Thus, full product, more strongly than embedded content, targets at equal main concepts.

The drop in Dice’s graph at position 68 to similarity value 0.375 is caused by a non-contiguous subexpression, five of whose six twigs do occur in the original. Full product, by its layer by layer computation, does not favour non-contiguous (layer skipping) subexpressions. A similar argument applies to the drops at positions 133, 214 and 263. Twig based measures thus are more robust for slightly differing non-contiguous subexpressions than full product.

Dice’s drop to the x-axis at position 271 is caused by the single term workshop which consists of the head of the original index expression only. As stated before, single terms have no twigs and Dice’s twig measure thus returns value zero.

Embedded Content

The second experiment imposes the embedded content order onto the full product and Dice measure for twigs. The results are given in Figure 4.17. Consider the line of the embedded content measure. A striking feature is the large rightmost plateau, stemming from the following. We computed the embedment of the original index expression in its subexpressions, not the other way around since this would result in maximal embedment for all subexpressions. Thus, the embedding of the original is computed in (the components
of) the subexpressions. For many subexpressions, this eventually results in the embedding of the original in the single terms of the subexpression. This, in turn, results in a constant value which is inversely proportional to the size of the original. As a consequence, embedded content is in this context best used for discriminating between highly similar subexpressions.

![Figure 4.17: Embedded content order imposed.](image)

In general, the order imposed by embedded content is relative to the removal of terms, irrespective of their depth. For instance, removing *Amsterdam* or *disk* results in equally similar subexpressions. Thus, embedded content makes no distinction to the specificity (depth) of the removed term. As a consequence, it is not so sensitive for the removal or addition of a complete modification (subexpression). This makes the embedded content measure suitable for matching queries with rich document characterisations.

After position 88, the graphs of full product and Dice-twigs seem to equal their original graphs of Figure 4.14(b). This is caused by the large dominant plateau, starting at that position, which, by resorting the subordinate measures, ensures that all the subexpressions are in the same order as in their original graph. Differences with their original graphs are caused by subexpressions that are placed in the last interval by the dominant measure but not by the original order of the subordinate measure, or vice versa.

A number of stepwise drops occur in the line of full product in Figure 4.17. For instance, full product drops to zero at positions 22, 23, and 84-87. This is caused by subexpressions with different heads. As was also observed for the first experiment, embedded content is more robust to changes in the main concept. The downward steps towards value zero are caused by subexpressions that miss several terms, but preserve other parts of subexpressions.

Dice’s twig measure shows a clear drop at position 65, which is caused by a (rather small) non-contiguous subexpression which has only a few twigs in common with the original. However, embedded content, being more robust to non-contiguous subexpressions, ranks it relatively high. A similar argument holds for the drops at positions 22 and 87. Thus, highly non-contiguous subexpressions are still recognised as relevant by embedded content.
Set-based Twig Similarity

In the final experiment, we compared set-based approaches for terms with those for twigs. In particular, the Dice measure for terms is used as baseline for comparison with the Dice, Jaccard, and Cosine measures for twigs. Figure 4.18 provides an overview of the used set-based measures, each measure being non-increasingly ordered as was done for the other measures in Figure 4.14(b).

Figure 4.18: Set-based approaches for terms and twigs.

In our experiments, twig-based measures have finer granularity than term-based measures. This is concluded from the number of intervals imposed by these measures. The set-based measures divide the subexpressions in classes (intervals) corresponding to the number of constituents in common with the original. As can be seen from Figure 4.18, the Dice measure for terms has 10 intervals for $\alpha = 1.0$ (i.e. no connectors considered) and 20 for $\alpha = 0.5$. The measures for twigs each have 26 intervals. This number directly depends on the number of different twigs that can be generated from the subexpressions. Since we included non-contiguous subexpressions in our experiment, new twigs are introduced that are not in the original. The original index expression has 10 twigs thus leaving room for 16 newly introduced twigs. This suggests that twig-based measures should be favoured over term-based measures if delicate matching is required.

The lines for the twig-based measures eventually drop to zero whereas the term-based measure does not. The reason for this is that single terms do not result in twigs although every (non-empty) subexpression has a minimal overlap of at least one term with the original. Clearly, twig-based measures should thus not be used for matching single terms. The minimal similarity value according to Dice for terms (for $\alpha = 1.0$) is $\frac{2x1}{11+1} = 0.167$.

In order to further compare the set-based approaches, the Jaccard ordering for twigs was imposed on the others. Figure 4.19 shows the results. Dice and Jaccard for twigs impose the same ordering: they have exactly the same intervals. As can thus be concluded from the definitions of these measures (see section 4.6.4), in comparing two subexpressions, no
distinction (in terms of the ordering) is made between the sum of twigs and the number of different twigs. Their similarity values, however, differ in size. This basically means that, practically, the differences between both measures can be overcome by using rightly set thresholds. The Cosine measure for twigs has closely resembling intervals: of its 26 intervals, only six differ slightly (maximally nine subexpressions).

There is a big difference between set-based approaches for terms and those for twigs, since terms do not capture structure. The peaks in the line of Dice’s term measure (Figure 4.19) are caused by non-contiguous subexpressions, not having many twigs in common. Irrespective of their structure, however, they share terms with the original. Clearly, when the structure of descriptors is to be taken into account, twig based measures are more suitable than term based measures.

The experiments illustrated characteristics of the different similarity measures. A number of conclusions can be drawn. First, approaches only using terms should be avoided if the structure of descriptors is relevant. This is what one would expect, as well as that set based measures using terms provide less granularity than those using twigs. Second, the different criteria about subexpressions resulted in similarity measures with distinct properties. The layer by layer computation of the full product measure, derived from the idea of headedness, favours descriptors that only differ in their most specific subexpressions. However inadequate for comparing index expressions that represent different main concepts, it naturally supports delicate discrimination of most subexpressions and a gradual notion of concept modification. The embedded content measure is not sensitive to the depth of the deleted subexpression. Furthermore, since it is based on embedding, it is more robust to non-contiguous subexpressions. Embedded content is, in the context of our experiments, best used for discriminating highly similar subexpressions. Twig based measures fall somewhere in between. They are somewhat sensitive to non-contiguous subexpressions: this introduces new twigs, but may preserve other twigs. There is little difference between twig based measures themselves. It appears mainly a matter of thresholds, viz size of the similarity values,
4.7. ADDITIONAL TOPICS CONCERNING IES

to distinguish between them.

4.6.6 Outlook on Matching

Future research can be directed towards augmenting index expressions with wildcards and the design of corresponding matching functions. This could aim at using the structure of index expressions for fact finding and the generation of partially specified overviews. In the augmented language, descriptors like flying from Tasmania to * can be used to find all documents that deal with air transport from the island of Tasmania to any destination.

Furthermore, it would be interesting to see if our approach is also viable for other retrieval related tasks such as clustering, routing, and filtering. This would enable an augmentation of the functionality of the INN system. For instance, the INN system could then present grouped results or it could work off-line by using stable information needs in the case of filtering.

It is also interesting to investigate different instantiations of the abstract term and connector comparison functions. For instance, stemming could be included by computing the similarity of stems. In addition, connectors can be grouped by their function type. In matching, connectors with the same function could be given higher similarity.

Including term weights potentially increases the effectiveness of matching. We did not focus on this issue here since it is not primarily concerned with the structure of index expressions. However, it seems a valuable extension worth further research.

4.7 Additional Topics concerning IEs

4.7.1 Relation with Noun Phrases

In [AWKB00b], a representation of noun phrases is given. It eliminates structures that can be assumed not to be beneficial for IR. Considering this representation of noun phrases, we sketch the relation with index expressions.

The representation of a noun phrase consists of determiners, a head, and pre- and post-modifiers. Determiners, such as articles and quantors, are not incorporated in index expressions. The head of noun phrases usually is a noun. This adheres to index expressions. Pre-modifying adjectives may be translated to subexpressions in index expressions, as described in [Ber98]. Some pre-modifiers, such as those formed by coordinated conjunctions, cannot be adequately modeled in index expressions. (The augmentations described in the following chapter do allow for such constructs.) Post-modifying constructs can be translated to nested subexpressions. Prepositional phrases, for example, can be translated to connector-subexpression pairs. Relative clauses that contain verb phrases may be harder to translate. This may be attacked by translating verb phrases into noun phrases by normalisation of the verb ([AWKB00b]). Concluding, there is a substantial part of noun phrases
that can be readily translated to IEs. A more complicated problem appears the extraction of correct noun phrases from text.

### 4.7.2 Obtaining Index Expressions from Text

As discussed in chapter 3, within the PROFILE project a special component has been assigned the task of linguistic analysis of text. For reasons of completeness, we elaborate on analysing text in order to obtain index expressions.

A simple parsing method based on a two-level priority scheme for connectors ([Bru93, BB91]) is used. All allowed connectors are listed together with their priority. All other words encountered are interpreted as terms (keywords). A stoplist (list of stopwords) filters out unwanted words like, for instance, articles and determiners. No additional sources of linguistic knowledge are used in parsing. During parsing, the priority of the connector at hand determines if the already parsed part of the index expression has to be broadened or deepened. Thus, no (explicit) linguistic grammar is used by the parser. The parser results in one parse tree only, thus possibly neglecting different interpretations. A test on the CACM document collection showed that this parsing method was accurate in approximately 90 percent of the cases.

A more general approach to indexing documents with index expression could make use of a part-of-speech (POS) tagger and an extractor. For closely related descriptors, called phrase frames, this approach is described in ([ATKW98]). Phrase frames are head-modifier structures in which connectors are (eventually) omitted. After preprocessing the documents, filtering non-textual parts and moulding the text into acceptable format, the POS tagger labels words with their part of speech. A well performing POS tagger is described in [Bri94]. The extractor, which is applied next, uses the POS labels in identifying the linguistic constructs that should be parsed as index expressions. The extractor uses pattern matching with regular expressions. An extractor for index expressions would constitute a subset of the patterns used for phrase frames. Finally, the same parser as above can be used, although more sophisticated parsers are available. This is, however, not in the scope of this thesis.

The general architecture of an NLP-IR system advocated by [Str95] also includes a stemmer. Normalisation at term level (e.g. stemming) is largely orthogonal to our approach, so possible to include, but not considered in this thesis. However, we provided abstract similarity functions for terms and connectors that could be instantiated by measures that do take, for instance, stemming into account.

Regularisation or syntactical normalisation at phrase level aims at recall enhancement by reducing syntactical variation ([Str95]). For phrase frames, syntactic normalisation is described in ([ATKW98]). This approach can, mutatis mutandis, also be applied for index expressions since phrase frames closely resemble index expressions. However, this requires the use of additional linguistic knowledge. We do not elaborate on this issue here.

Instead of normalising descriptors, systematically generating (all) linguistically motivated alternative variants ([SJTW84]) has a similar aim. Semantic linguistic knowledge is exploited
not to generate invalid variants. Although this restricts the number of generated variants, this number may explode, especially for complex composed descriptors. Matching is done via translation of the generated variants into a Boolean query. Due to the need for efficient matching in the INN system, we do not generate all variants of index expressions.

The Constituent Object Parser ([MH89]) also uses syntactic structure for matching descriptors. It produces binary dependency trees, which indicate (directed) dependency and scope of relationships between terms. Many different syntactic forms are represented by this single dependency relation. As a result, ambiguities are retained in the representation ([Str95]). A comparable approach is that of using tree structured analytics ([SS91]). Both approaches use external sources of linguistic knowledge and offer at best reasonable efficiency. Index expressions make a distinction between the “types” of dependency, as given by connectors. Not taking the actual (contents of) connectors into account, e.g. by allowing wildcard connectors, in matching normalised index expressions effectuates a similar (directed) notion of dependency. An intermediate form of dependency, colliding connectors of the same type, is described in section 4.6.3.

The CLARIT system ([EGWH+91, ELG+92]) integrates several NLP techniques with the vector space retrieval model. These techniques include morphological analysis, robust noun-phrase parsing, and automatic construction of first order thesauri. Construction of the thesauri, used to support the selection of appropriate terms, requires a substantial (minimally 2 GB) sample of documents about a certain topic. Because of limited response times and available memory, we expect this approach too demanding for our application.

### 4.7.3 Counting Index Expressions

In this section, we devise a function that counts the number of all possible index expressions with a given number of terms. The number of connectors in an index expression is one less than the number of terms in it. Therefore, the number of connectors could also have been used as criterion.

![Figure 4.20: Sketch of Computation.](image)

As Figure 4.20 sketches, index expressions can be initially classified by the degree of branching of the root. In this way, a partitioning is made containing index expressions that have the same branching degree in the root. Counting index expressions then comes down to adding the sizes of these individual classes. There are \( t - 1 \) such classes since branching degrees at the root range from one to \( t - 1 \). The first class in Figure 4.20 has branching degree one, placing all terms but the root in a single subexpression. The second class has a branching degree of two and divides the remaining terms in its two subexpressions. The
final class has \( t - 1 \) subexpressions directly under the root. All of these consist of a single term.

The expression \( R(t) \) denotes the number of all possible index expressions with \( t \) terms. It is computed as the sum of the sizes of all possible classes of index expressions with \( t \) terms. The sizes of classes are given by \( R(t') \), where \( t' \) is smaller than \( t \). This enables \( R \) to be defined as a recurrent function. This is done in equation (4.8).

\[
R(t) = R(t - 1) + \sum_{t_1 + t_2 = t - 1} R(t_1)R(t_2) + \sum_{t_1 + t_2 + t_3 = t - 1} R(t_1)R(t_2)R(t_3) + \ldots + \sum_{t_1 + t_2 + \ldots + t_{t-1} = t - 1} R(t_1)R(t_2) \ldots R(t_{t-1})
\]  

(4.8)

In equation (4.8), \( R \) is recursively defined in terms of all possible classes with different branching degrees at the root. The expression \( R(t - 1) \) corresponds to the number of index expressions in the class with branching degree one. On the second line, \( \sum_{n_1 + n_2 = n - 1} R(n_1)R(n_2) \) denotes the number of index expressions in the class with branching degree two. It is computed by summing over all possible ways to divide the remaining \( t - 1 \) terms over the two subexpressions. For every combination of \( t_1 \) and \( t_2 \) such that \( t_1 + t_2 = t - 1 \), one of the subexpressions receives \( t_1 \) terms and the other the remaining \( t_2 \). By multiplying the number of possible index expressions for the two subexpressions, i.e. \( R(t_1) \) and \( R(t_2) \), the total number of index expressions with branching degree two is obtained. This process is repeated for every possible branching degree and the results are summed.

The computation in equation (4.8) involves splitting the \( t - 1 \) terms in every possible series of smaller \( t \)-values. That is, it involves computing the partition of \( t - 1 \). We may thus express \( R \) as done in equation (4.9).

\[
R(n) = \sum_{\pi \in \text{Partition}(n-1)} \prod_{i=1}^{\pi} R(n_i)
\]

(4.9)

Figure 4.21 illustrates the rapid growth of the number of index expressions as the number of terms increases. The y-axis is in logarithmic scale and denotes the number of possible index expressions. On the x-axis, the number of terms constituting the index expressions are placed.

The computation of the number of index expressions as described above involves a normalisation with respect to contents. That is, it assumes that all terms and connectors are equal. Thus, it only computes possible different structures. All such structures that can be made with one, two, three, and four terms are depicted in Figure 4.22.

\[
t(n) = \frac{1}{n} \left( \frac{2(n - 1)}{n - 1} \right)
\]

(4.10)
4.8. OUTLOOK

Logarithmic Growth of Number of Index Expressions

Figure 4.21: Logarithmic growth in possibilities.

As derived in [Knu75], the number of ordered rooted trees is given by equation 4.10. This number equals the number of possible structures for index expressions $R(n)$, as specified in equation 4.9.

4.8 Outlook

Currently, the WWW is a popular information space. Its size and dynamics, however, divert from the classical document domains considered in IR. Therefore, the application of navigational networks of IEs in large and dynamic information spaces deserves further attention. This topic is treated in chapter 6.

The social aspect of searching information becomes more prominent. This means that users are not seen as unconnected individuals, but as members of some community. More elaborate hypertext structures seem appropriate for this task, combining information about several users and document content. This is described in section 6.7.

The language of index expressions offers restricted ways of concept construction. A research question concerns augmenting the language of IEs in useful ways. In chapter 5, logical operators are added to IEs, providing an expressive and compact representation.
Chapter 5

Boolean Index Expressions

5.1 Introduction

The first chapter of this thesis provided three important properties of descriptors: expressiveness, tractability, and compactness. As the previous chapter showed, index expressions go beyond the bag-of-words model by including linguistically motivated structure. However, index expressions do not allow compact representations and do not provide logical concept construction. Boolean operators offer the expressive power that can be used to obtain compact descriptors.

In this chapter, a descriptor language is presented that combines linguistically motivated structure with Boolean operators. This descriptor language, called Boolean index expressions (BIEs), presents a workable tradeoff between the three properties. It allows complex information needs to be compactly represented. For use in ID, it is not required to index documents with BIEs. Instead, it is shown how arbitrary BIEs can be matched with (sets of) index expressions.

The structure of this chapter is the following. In section 5.2, BIEs are introduced and mechanisms for constructing them are illustrated. Section 5.3 elaborates on the compactness of BIEs, relating it to the notion of expansion. Section 5.4 describes the way to zip BIEs into normal form which may expand them. Section 5.5 formally introduces a measure of expansion. Minimal and maximal expansion are examined in sections 5.6 and 5.7, respectively. Section 5.8 illustrates the use of BIEs in ID. Section 5.9 provides an implementation of BIEs in a functional language. Section 5.10 summarizes the conclusions and provides directions for further research.

5.2 Boolean Index Expressions

Boolean index expressions (BIEs) form an expressive synthesis of linguistically motivated modification from index expressions and logically motivated concept construction from the
Boolean model. This is schematically depicted in Figure 5.1. As can be seen from this figure, BIEs are not as complex as noun phrases, thus offering better tractability. Furthermore, BIEs are designed to include nested logical operators, enabling high compactness.

![Figure 5.1: Two types of structure in BIEs.](image)

It should be remarked that the terms logical and linguistic structure have many interpretations. In Figure 5.1, we restrict ourselves to illustrating the two mechanisms for constructing BIEs. Furthermore, we acknowledge that this figure presents a simplification of noun phrases.

### 5.2.1 Representation of BIEs

BIEs are constructed on the basis of terms, connectors, the structural operator `add`, and the logical operators for disjunction (`∨`), conjunction (`∧`), and negation (`¬`). The language of BIEs is given by the following inductive definition, which is taken from [WBW98c], allowing for structural induction on BIEs.

**Definition 5.1 Boolean Index Expressions** Let $T$ be a set of terms and $C$ be a set of connectors. The language of Boolean index expressions, denoted by $\mathcal{L}^B_{(T,C)}$, is inductively defined by

- **Term** if $t \in T$ is a term, then it is also a Boolean index expression,

- **Add** if $I$ and $J$ are Boolean index expressions and $c \in C$ is a connector, then $\text{add}(I, c, J)$ is a Boolean index expression,

- **Dis, Con, Neg** if $I$ and $J$ are Boolean index expressions, then $I \lor J$, $I \land J$, and $\neg I$ are Boolean index expressions as well, and
Stop no other Boolean index expressions exist.

The language of BIEs includes the (Boolean) keyword language, the language of index expressions, and logical combinations thereof. This is stated in the following two lemma’s. Boolean combinations are formed by disjunctions, conjunctions, and negations.

**Lemma 5.1** Each (Boolean combination of) keyword(s) is a BIE.

**Lemma 5.2** Each (Boolean combination of) index expression(s) is a BIE.

Lemma’s 5.2 and 5.1 are illustrated below. For reasons of readability, the structural operator is not always represented explicitly.

**Example 5.1** Lemma 5.1 is illustrated by conference $\land$ ICT $\lor$ $\neg$ Holland. Lemma 5.2 is illustrated by the following BIE $\text{add}$(conference, on, ICT) $\land$ add(conference, in, Holland).

BIEs introduce descriptors that do not belong to the included languages. Such descriptors contain nested operators, such as, for example, conference on (ICT $\land$ Agents) and conference on $\neg$ ICT.

For practical purposes, grammar representations are helpful. Figure 5.2 provides a grammar for BIEs. In this grammar, NBoolExp stands for non-empty Boolean index expression. The primed forms ensure that priorities of the logical operators are correctly parsed. This is explained in Section 5.4.1. The grammar is suitable for agents that receive BIEs as part of incoming messages and require them to be parsed, translating BIEs in textual representation to tree structures. Note, however, that this grammar is not meant to parse full text documents.

**Figure 5.2:** Concrete Grammar of Boolean Index Expressions.
CHAPTER 5. BOOLEAN INDEX EXPRESSIONS

5.2.2 Properties of BIEs

BIEs may be used as descriptor language for retrieval and filtering of information. As identified in the introduction, the following properties of descriptor languages are important: expressiveness, tractability, comprehensibility, and compactness. BIEs make a realistic trade-off between the three mentioned criteria. These criteria are discussed below.

**Expressiveness** BIEs are more expressive than (logical combinations of) keywords (lemma 5.1) and (logical combinations of) index expressions (lemma 5.2). Relative to logical combinations of index expressions, nested occurrences of logical operators ensure extra expressiveness. This was illustrated in example 5.1. BIEs are less expressive than noun phrases, since BIEs are an abstraction of noun phrases. As stated before, however, noun phrases may be too expressive for retrieval purposes ([Sme97]).

**Tractability** BIEs correspond to an easily tractable part of (linguistic) noun phrases. Therefore, noun phrases are less tractable than BIEs. However, not every BIE can be efficiently processed. Zipping, for example, may result in BIEs of impracticable size. Still, in section 5.7.4, we show that for application in information retrieval and filtering, BIEs are sufficiently tractable. Because of their simplicity, keywords and index expressions are well tractable. This also holds for Boolean keywords, where the Boolean operators allow high standards of performance ([SFW83]) by enabling bitwise implementation of matching.

For practical applications, efficient operations can be devised on BIEs. Examples of such operations are matching by similarity functions, transformation in normal form, equivalence checks, and (partial) evaluation of the logical operations.

**Compactness** BIEs offer a compact representation of informational content. With a limited number of terms, connectors, and operators, much information can be conveyed. The reason for this is that Boolean operators can occur nested in the structure imposed by connectors, allowing BIEs to be used as concept representation. Nested dyadic operators effectuate subexpression sharing, requiring the shared subexpressions to be represented only once. Note that in, conference on (ICT ∧ engineering), for example, the conference on-part is shared by ICT and engineering. The bag-of-words model and index expressions are least compact, since they offer no means for subexpression sharing. Boolean keywords offer some compactness by nested logical operators. For example, the logical combination of keywords conference ∧ (ICT ∨ engineering) can be expanded to (conference ∧ ICT) ∨ (conference ∧ engineering). The lack of linguistically motivated nesting, however, restricts this compactness. Although noun phrases potentially offer high compactness, current linguistic techniques are not properly capable of extracting compact noun phrases from text. Compact BIEs can be constructed in several ways, which is illustrated in section 5.2.3. In later sections of this chapter, compactness of BIEs is investigated in detail.
Concluding, BIEs form a workable tradeoff between the four mentioned criteria, enabling compact representations of information needs while maintaining practical applicability in information retrieval and filtering.

The semantics of BIEs depends on two issues. First, since BIEs are based on index expressions, their nature is partly syntactic. Second, BIEs contain logical operators of which the semantics is partly specified.

Zipping, as described in the next section, partly defines the meaning of the Boolean operators. For instance, the meaning of nested dyadic operators is specified in the context of connectors. In the intended semantics, disjunctions have lower priority than conjunctions ([WBW98c]). Note that this conforms to the familiar semantics known from propositional logic. In addition, zipping exploits (generalisations of) transformation rules from propositional logic.

The semantics of logical operators in BIEs is not completely defined. Double negations, for example, are not crossed out against each other. This leaves room for different views on their semantics, if required. For instance, one may choose to adopt the closed world assumption ([Rei78]) or implement negations as dyadic operators by using an ontology.

The lack of linguistic semantics and the high degree of abstraction of BIEs makes that certain linguistic issues cannot be coped with properly. Examples of such issues are homonymy and lexico-semantical variation. We argue, however, that an indirect semantics defined via a collection of documents is suitable for many IR and IF tasks. Furthermore, a high degree of abstraction combined with compactness, comprehensibility and expressiveness makes BIEs suitable for representing information needs.

The following subclass of BIEs forms a special class.

**Definition 5.2 Atomic BIEs**

A BIE is atomic, denoted by $\text{Atomic}(I)$, iff it does not contain disjunctions or conjunctions. In other words, if it is constructed from terms, composition (add), and negations.

Atomic BIEs are relevant, because they form the basic building blocks of normalised BIEs, as shown in section 5.4.

### 5.2.3 BIEs for Retrieval and Filtering

This section illustrates the formulation of BIEs in IR. In IR, it is of great importance that users formulate their information need concisely. However, formulating an information need with complex descriptors is a difficult task. Therefore, the semantics of Boolean operators may not be coherently interpreted by different users. Therefore, we describe two ways to support the user in formulating his information need with BIEs. First, a profile and query constructor tool is illustrated providing an integrated approach to searching and exploration. Second, the well-known mechanism of relevance feedback is used for descriptor reformulation.
Profile and Query Constructor Tool

An envisaged profile and query constructor tool can support the user during formulation of his information need. In this section, the rationale behind this tool is elaborated.

![Diagram of Profile and Query Constructor Tool](image)

**Figure 5.3: Interface of Profile and Query Constructor Tool.**

The interface of the constructor tool (see Figure 5.3) is divided into four windows. Together, they provide an integrated approach to formulating information needs. The left upper window shows the BIE under construction. For maximal comprehensibility, both a graphical and an alpha-numerical representation are provided. A part of the BIE, say the term *retrieval*, may be selected as starting point for navigational exploration in an overview structure, called the hyperindex. Therefore, the right upper window gives a relevant fraction of the hyperindex associated with the document collection, providing a graphical overview of the domain directly surrounding the selected part. In this example, the searcher has apparently move in the hyperindex from *retrieval* to *information retrieval*, and might be willing to refine the expressions *retrieval* by this more specific alternative. Descriptors found by navigation may be included in the left upper window. Hyperindices for index expressions exploit the availability of linguistically motivated subexpressions in index expressions to enable structured navigation. In the left lower corner, a set of possible actions for modifying the BIE is provided. These constitute actions for refining the BIE, including synonyms, forming phrases, excluding topics, normalising the BIE under construction, and deleting a part of the BIE. The textual representation may be directly modified by a syntax-based editor. It is envisaged that synonyms and phrases can be formulated by navigation in the hyperindex. In the right lower corner, a ranked list of documents relevant to the constructed BIE is given. In this way, the user directly gains insight in the effects of the modifications made to the query or profile. Note that this indirect semantics is domain sensitive.

Visualising BIEs assists the user in understanding their meaning. BIEs can be visualised
as nested graphs, graphs in which the nodes may be graphs themselves. An example of this is given in the left upper corner of Figure 5.3 visualising the BIE (retrieval OR filtering) of (text AND images). An elaborate overview of graph visualisation and navigation techniques is given in [HMM99].

This tool enables formulation of information needs by combining searching and browsing in a hypertext environment. This combined approach to IR (advocated in e.g. [HPW96, WC91, LZ93, BW92]) eases some well-known problems in formulating information needs such as dynamic and vague information needs, too broad queries, and syntactical correctness and sensibility of descriptors. Searching is effectuated by directly modifying the BIE. Browsing can be done by navigating in the hyperindex. Furthermore, by inspecting the documents that are relevant to nodes in the hyperindex, the user’s knowledge about the topic of interest increases.

Relevance Feedback for Constructing BIEs

In IR, relevance feedback is a well-known technique for query reformulation. Upon rendering of documents for an initial query, the user is enabled to explicitly mark (parts of) documents as relevant or irrelevant. This is called positive and negative relevance feedback, respectively. Based on this feedback, the initial query is refined after which it is fired off to the retrieval system for another iteration. This process terminates if the information need is satisfied.

Assume an initial query is formulated as a BIE. Using similarity measures for BIEs ([WBW98c]), the query can be matched with available documents. As a result, a ranked list of documents is produced. These documents are characterised by index expressions. Index expressions can be automatically extracted from text with sufficient precision ([Bru93, ATKW98]). Current natural language processing techniques do not (yet) support this for BIEs. Relevance feedback can be supported by enabling the user to mark individual index expressions or complete document characterisations as relevant or irrelevant by simple mouse clicks.

![Diagram of a tree-like structure representing a BIE]

Figure 5.4: Relevance Feedback for Constructing BIEs.

The marked index expressions are used to refine the initial query. All positively identified index expressions are added to the initial query in conjunction. The negation of all nega-
tively identified index expressions are added as well. Negatively identified index expressions are included in the “AND NOT” way, ruling out unwanted concepts. The resulting BIE, schematically depicted in Figure 5.4, can thus be divided in three parts: (1) the initial query, which may contain nested logical operators, (2) a part for the positively identified index expressions, and (3) the negatively identified part. The latter two parts may, of course, contain more than two subexpressions. For presentational reasons, the topmost conjunction is depicted as a tertiary operator. This illustrates that different graphical representations may be suitable.

An advantage of the formulation technique based on relevance feedback is control of the form of the query. The index expressions obtained from relevance feedback, i.e. these positively and negatively identified, do not contain logical operators. Therefore, the two rightmost parts of the BIE of Figure 5.4 can be processed with techniques for index expressions.

5.3 On Compactness of BIEs

There are many different ways to express the same information with BIEs. That is, syntactically different BIEs may be semantically equivalent. Therefore, within equivalence classes of descriptors, different levels of compactness can be reached by utilising overlap in concepts. This is sketched in Figure 5.5. Minimal compactness is reached for combinations of elementary descriptors, not utilising overlap between the elementary descriptors to form a compact representation. Thus, minimally compact BIEs may require many constituents to express complex information needs. Thus, for complex needs, they are not economical for presentation to the user on screen or for storage. However, an advantageous feature of minimally compact BIEs is that their semantics can be expressed in terms of elementary descriptors. In addition, minimally compact BIEs may be easily processed because of their standardised structure in which the priorities of logical operators are not ambiguated by connectors.

At the other end of the compactness-continuum, overlap in analogous concepts is fully exploited by representing the shared concept just once. This yields maximally compact descriptors carrying several refinements of shared subexpressions. From a formulation point

![Diagram of equivalence class of BIEs.](image)
of view, compactness allows a high degree of integration, combining several occurrences of an analogous concept in a single expression. Compact BIEs may be economically presented and stored. However, processing compact BIEs is more difficult since their structure does not feature elementary subexpressions. The reason for this is that the logical operators in maximal BIEs are nested within the linguistically motivated refinement structure. Furthermore, it may be difficult to bring arbitrary BIEs in their most compact representation.

Insight in the nature of compactness supports the design of applications with BIEs. It is thus valuable to know what minimally and maximally compact BIEs look like. For instance, explicitly characterising such BIEs eliminates the need to zip them in order to decide if they are minimal or maximal. This is a valuable property since zipping may be costly. In query formulation tools, for instance, the system may exploit the explicit characterisations to lead the user to specific types of BIEs, taking into account system criteria (e.g., efficiency of processing) or user criteria (e.g., required space for presentation).

In addition, explicitly characterising minimally and maximally expanding BIEs opens doors for experimental quantitative analyses since it eases the design of efficient generator and test functions.

A measure for compactness of BIEs can be based on normalisation, which reduces their syntactical variance. We provide a normalisation function that maps BIEs to a semantically equivalent BIE into normal form. This translation involves an operator called zipping which unnests the nested dyadic operators and brings BIEs in a special (disjunctive) normal form. The normal form may require a larger number of operators, terms, and connectors. This effect is called expansion. We show that expansion is directly proportional to compactness. Therefore, expansion forms a vehicle for studying compactness.

Compactness, and thus expansion, should be quantified in order to ascertain their asset. Therefore, we examine quantitative aspects of expansion. For instance, we provide tight bounds on the expansion of BIEs by examining the expansion of minimally and maximally compact BIEs. Another quantitative issue concerns the fractions of BIEs that are minimal and maximal, respectively. For instance, a positive feature of a class of BIEs may be of little asset if such BIEs rarely occur. We research these questions both theoretically and experimentally.

Concluding, one of the goals of this chapter is to investigate the nature and quantitative aspects of compactness of BIEs via their expansion. In particular, minimally and maximally expanding BIEs are examined.

### 5.4 Zipping Operators in BIEs

As stated before, compactness is an important property of BIEs for use in IR and IF. A compact representation is obtained via nested dyadic operators and subexpression sharing. Compactness is studied via a normal form for BIEs, which is obtained by a transformation that unnests nested dyadic operators. The transformation, called *zipping*, transforms BIEs in semantically equivalent BIEs in normal form. Generally, the BIE in normal form requires
more operators, terms, and connectors than the original one. The increase in required operators is called expansion. Thus, we study compactness via expansion by zipping.

In this section, we first illustrate the intended semantics of BIEs (section 5.4.1) and describe a general format for normal forms, called the propositional form (section 5.4.2). In section 5.4.3, we provide a constructive description of zipping. Then, section 5.4.4 shows that the result of zipping is in disjunctive normal form, which is a special case of propositional form. Finally, section 5.4.5 provides a number of auxiliary lemma's that are needed in characterising and measuring expansion.

### 5.4.1 Priority Scheme

The intended semantics of BIEs are expressed by breaking them down into a propositional formula constructed from atomic BIEs. The semantics of BIEs that do not contain nested occurrences of the logical operators is provided first. Then, the mutual influence of (logical) operators and (linguistic) connectors is investigated.

First, we concentrate on BIEs without nested logical operators. The semantics of purely logical combinations of index expressions adheres to the standard semantics of the logical operators. These rules only deal with logical operators, not with their combination with linguistic connectors.

Second, we investigate the semantics of BIEs in which logical operators occur nested with respect to linguistic connectors. That is, we investigate the mutual influence of (logical) operators and (linguistic) connectors. First, disjunctions and conjunctions are examined, followed by an investigation of negations.

Nesting logical operators with connectors may result in ambiguity. This ambiguity occurs when conjunctions and disjunctions co-occur in a Boolean index expression and are connected through at least one connector. The ambiguity involves the order of evaluation, or priorities, of the logical operators. As an example, consider the sentence

\[
\text{walking with friends and relatives to Holland or Germany}
\]

This sentence could alternatively be expressed as

\[
\begin{align*}
\text{walking with friends and relatives to Holland} \\
\text{or} \\
\text{walking with friends and relatives to Germany}
\end{align*}
\]

by first concentrating on the or-operator. First concentrating on the and-operator leads to

\[
\begin{align*}
\text{walking with friends to Holland or Germany} \\
\text{and} \\
\text{walking with relatives to Holland or Germany}
\end{align*}
\]

The above alternative representations hinge on different priorities for dyadic operators. From an engineering point of view, we are forced to choose between two different normal
forms. Based on our intuition, we adopt the first interpretation, meaning that (as usual) the and-operator has a higher priority than the or-operator. As an illustration, note that the second interpretation would allow a document dealing with

walking with friends to Holland and walking with relatives to Germany

to be relevant to sentence (5.1). However, the intuition behind sentence (5.1) seems to exclude this as a possible interpretation.

These priorities conform to propositional logic ([Weg87]). A small evaluation amongst fellow researchers and students indicated that the majority (about 65%) intuitively supported the priorities as defined. The evaluation considered minimal examples in which both a disjunction and a conjunction occurred, separated by a connector. This does not, however, mean that other priority schemes (binding schemes) are not applicable, that there are no intuitive counterexamples, or, even, that dyadic operators always distribute over connectors. Despite these considerations, we argue that our definition is reasonable and will show its benefits in the remainder of this chapter.

Unlike dyadic operators, the unary negation operator is defined not to distribute over connectors. This is illustrated in the next example.

**Example 5.2** The grammar of BIEs allows for four variants of negations in cooking for singles:

\[
\neg (\text{cooking \ for \ singles})
\]
\[
\neg (\text{cooking \ for \ singles})
\]
\[
\text{cooking \ for \ } \neg \text{ singles}
\]
\[
\neg (\text{cooking \ for \ } \neg \text{ singles})
\]

Our intuition suggests that, considered as a query, the first BIE is satisfied with every document that is not about cooking for singles. The second is satisfied by anything that is for singles, except cooking. The third is satisfied by documents about cooking for others than singles. Finally, the fourth is satisfied by documents that describe something other than cooking for something else than singles.

Example 5.2 shows that the combination of negations with connectors does not define patterns of equivalent Boolean index expressions. That is, no normalisations can take place that consist of lifting a negation over a connector. This means that negations can only be normalised in the context of other logical operators, which is done via the standard logical interpretation of negations.

### 5.4.2 Purpose of Zipping

BIEs are put into normal form by a process called zipping. By unnesting dyadic operators in BIEs, zipping pushes disjunctions and conjunctions upward. As a consequence, negations are zipped downward. This is illustrated by the following example.
Example 5.3 The BIE, walking with (friends AND relatives) to (Holland OR Germany), as shown graphically in Figure 5.6, contains a nested conjunction and a nested disjunction.

Zipping the BIE of Figure 5.6 results in that of figure 5.7. In this zipped BIE, the disjunction and conjunction are lifted out of the nested structure. Note that it contains more operators, terms, and connectors than the original. Furthermore, note that, according to the intended semantics, both BIEs are equivalent.

Other forms of zipping may aim at different results. A different form of zipping can be obtained by zipping the dyadic operators downward. As a consequence, negations will then be pushed upward. That form of zipping aims at a compact representation by grouping together shared components. In this chapter, however, we concentrate on zipping dyadic operators upward.

In the normal form obtained by zipping (see Figure 5.8), two parts can be distinguished. The topmost part, the so called logical part, forms the logical structure of the proposition. It consists of disjunctions and conjunctions only. The lower part, the atomic part, consists of atomic BIEs. This two-layered form is called the propositional form for BIEs.

An advantage of the propositional form is that it eases the implementation of operations on BIEs since the normalised structure allows the logical and atomic parts to be treated separately. The atomic part can be treated by (slightly altered) procedures for index expressions. Example operations are matching, computing the degree of similarity between two BIEs, and equivalence checks. In addition, the normal form can be exploited to express the semantics of BIEs in terms of a logical combination of atomic BIEs. Since atomic BIEs
do not contain dyadic operators they are of a simpler nature. For several classes of BIEs, the propositional form is a logical combination of index expressions. This further eases the design of operations.

A BIE in disjunctive normal form (DNF) is a logical sum of logical products of atomic BIEs.

**Definition 5.3** A BIE is in DNF iff it is of the form

\[ \bigvee_{1 \leq i < k} \bigwedge_{1 \leq j \leq l_i} I_{i,j} \]

where all \( I_{i,j} \) are atomic.

Similarly, a BIE in conjunctive normal form (CNF) is a logical product of logical sums of atomic BIEs. The priorities of dyadic operators favour the construction of formulae in DNF, since disjunctions are to be evaluated before conjunctions. In order to obtain an equivalent BIE in CNF, more work has to be done and more complex descriptors may be obtained. Therefore, our zipping procedure is designed to deliver equivalent BIEs in DNF.

### 5.4.3 Zipping Functions

In this section, we provide a functional description of normalisation for BIEs. That is, we provide several functions that perform zipping. An advantage of this approach is that the functions are readily translated to functional programs. This allows a simple implementation of BIEs for practical purposes. The PROFILE filtering system ([SAW+00]), for example, uses a functional implementation of BIEs. Other suitable formalisms exist. For example, conditional term rewriting systems (see e.g. [Klo92]) could alternatively be used for specifying zipping.

Bringing BIEs into propositional form can be done in several ways. The Zip function, as defined below, does so by explicitly taking the priorities of operators (as discussed in section...
5.4.2) into account. The priority of disjunctions is lower than that of conjunctions. This is
effectuated by zipping disjunctions $\text{ZipOr}$ before conjunctions $\text{ZipAnd}$.

$$\text{Zip} : \mathcal{L}_{(T,C)}^B \to \mathcal{L}_{(T,C)}^B$$

$$\text{Zip}(I) = \text{ZipAnd}(\text{ZipOr}(I))$$

In sections 5.4.3 and 5.4.3, we define the functions $\text{ZipOr}$ and $\text{ZipAnd}$. Both function have
the same type as $\text{Zip}$.

**Zipping Disjunctions**

Function $\text{ZipOr}$ promotes all disjunctions and, if necessary, some conjunctions.

$$\begin{align*}
\text{ZipOr}(t) &= t \\
\text{ZipOr}(\text{add}(I, c, J)) &= \text{OrCons}(\text{ZipOr}(I), c, \text{ZipOr}(J)) \\
\text{ZipOr}(I \lor J) &= \text{ZipOr}(I) \lor \text{ZipOr}(J) \\
\text{ZipOr}(I \land J) &= \text{OrProd}(\text{ZipOr}(I), \text{ZipOr}(J)) \\
\text{ZipOr}(\neg I) &= \text{NotProd}(\text{Zip}(I))
\end{align*}$$

Terms do not contain disjunctions and therefore remain unprocessed. For the composed
BIE $\text{add}(I, c, J)$, disjunctions in both $I$ and $J$ are elaborated recursively, resulting in their
disjunctive representations $\text{ZipOr}(I)$ and $\text{ZipOr}(J)$. From these, an overall disjunction is
constructed by $\text{OrCons}$. The function $\text{OrCons}$ distributes its disjunctive arguments via the
provided connector.

$$\text{OrCons} : \mathcal{L}_{(T,C)}^B \times C \times \mathcal{L}_{(T,C)}^B \to \mathcal{L}_{(T,C)}^B$$

$$\text{OrCons}( \bigvee_{1 \leq i \leq k} I_i, c, \bigvee_{1 \leq j \leq l} J_j) = \bigvee_{1 \leq i \leq k, 1 \leq j \leq l} \text{add}(I_i, c, J_j)$$

In the case of a disjunction $I \lor J$, the result of $\text{ZipOr}$ consists of the disjunction of the
disjunctive representations $\text{ZipOr}(I)$ and $\text{ZipOr}(J)$. In the case of a conjunction $I \land J$, another auxiliary routine, $\text{OrProd}$ is used to ensure that a disjunctive result is produced.
The function $\text{OrProd}$ is a generalisation of DeMorgan’s law. It distributes its disjunctive
arguments over a conjunction.

$$\text{OrProd} : \mathcal{L}_{(T,C)}^B \times \mathcal{L}_{(T,C)}^B \to \mathcal{L}_{(T,C)}^B$$
In the case of a negation $\neg I$, two steps have to be performed consecutively. First, expression $I$ is put into disjunctive normal form by a recursive application of the Zip operator. Second, function NotProd distributes the negation over the propositional part and restores the disjunctive normal form by a call to a generalised version of OrProd.

$$\text{OrProd}(\bigvee_{1 \leq i \leq k} I_i, \bigvee_{1 \leq j \leq l} J_j) = \bigvee_{1 \leq i \leq k, 1 \leq j \leq l} (I_i \land J_j)$$

Since the argument of NotProd results from the recursive call to Zip, it is in DNF. Distributing the negation over this BIE would yield a BIE in CNF, i.e. of the form

$$\bigwedge_{1 \leq i \leq k} \bigvee_{1 \leq j \leq l_i} \neg I_{i,j}$$

The $k$ arguments of this conjunctive structure are all passed to OrProd'. Function OrProd' can simultaneously deal with more than one conjunction, by being allowed more than two arguments. The generalised OrProd' is specified in terms of the dyadic function OrProd as follows

$$\text{OrProd}'(\bigvee_{1 \leq j \leq l_i} \neg I_{i,j}, \ldots, \bigvee_{1 \leq j \leq l_k} \neg I_{k,j}) = \text{OrProd}(\bigvee_{1 \leq j \leq l_i} I_{1,j}, \ldots, \text{OrProd}(\bigvee_{1 \leq j \leq l_{k-1}} I_{k-1,j}, \bigvee_{1 \leq j \leq l_k} I_{k,j}))$$

in case OrProd' has two or more arguments. In case it has only one argument, it returns the unchanged argument.

Function NotProd does not cancel double negations since imposing the closed world assumption (CWA) ([Rei78]) may have unwanted effects for retrieval. Including the CWA may result in poor precision because characterisations usually do not include topics documents are not about. Imposing the CWA also leads to a non-monotonic aboutness proof system ([Hui96]).

**Zipping Conjunctions**

After zipping all disjunctions, the remaining conjunctions are zipped by ZipAnd. The function ZipAnd is to be called after ZipOr since it assumes its argument is a logical sum of BIEs that do not contain disjunctions. The function ZipAnd zips conjunctions locally, i.e. without modifying the disjunctive structure that resulted from ZipOr.
\[
\begin{align*}
\text{ZipAnd}(t) &= t \\
\text{ZipAnd}(\text{add}(I, c, J)) &= \text{AndCons}(\text{ZipAnd}(I), c, \text{ZipAnd}(J)) \\
\text{ZipAnd}(I \lor J) &= \text{ZipAnd}(I) \lor \text{ZipAnd}(J) \\
\text{ZipAnd}(I \land J) &= \text{ZipAnd}(I) \land \text{ZipAnd}(J) \\
\text{ZipAnd}(\neg I) &= \neg I
\end{align*}
\]

Similar to ZipOr, the result of zipping a term is that term itself. For composed BIEs \(\text{add}(I, c, J)\), all conjunctions in \(I\) and \(J\) are zipped by recursive calls to ZipAnd. After that, the function AndCons constructs the final conjunction by distributing its conjunctive arguments via the provided connector.

\[
\text{AndCons} : \mathcal{L}_B^C \times C \times \mathcal{L}_B^C \to \mathcal{L}_B^C
\]

\[
\text{AndCons}(\bigwedge_{1 \leq i \leq k} I_i, c, \bigwedge_{1 \leq j \leq l} J_j) = \bigwedge_{1 \leq i \leq k, 1 \leq j \leq l} \text{add}(I_i, c, J_j)
\]

The disjunctive structure that resulted from ZipOr is not to be altered. Rather, only in the non-disjunctive arguments of the disjunction are conjunctions allowed to be zipped. Therefore, the ZipAnds are recursively placed in the arguments of the disjunction without altering the disjunctive structure.

Conjunctions \(I \land J\) are rewritten to the conjunction of the zipped versions of \(I\) and \(J\).

For ZipAnd, zipping a negation \(\neg I\) is simpler than for ZipOr. This is because after ZipOr has been applied, \(I\) must be atomic. Thus, in case ZipAnd is applied to a negation \(\neg I\), \(I\) does not contain disjunctions or conjunctions and is thus already in DNF.

### 5.4.4 Result of Zipping

Zipping reduces the expressive variety in BIEs by bringing them into propositional form. In this section, we show that the two-phase zipping procedure described in the previous section yields BIEs in disjunctive normal form. Two definitions and a result on OrProd are required to state the DNF lemma.

Stating that zipping a certain BIE does not result in a disjunction must take the inverting effect of negations into account. During zipping, negations turn disjunctions into conjunctions and vice versa by application of NotProd. The inverting effect of negations is reflected in the definition below by alternating between disjunctions and conjunctions when a negation is encountered.
5.4. ZIPPING OPERATORS IN BIES

**Definition 5.4** We say a BIE $I$ contains $\text{DisCount}(I)$ disjunctions and contains $\text{ConCount}(I)$ conjunctions, where $\text{DisCount}$ and $\text{ConCount}$ are defined as

\[
\begin{align*}
\text{DisCount}(t) & = 0 \\
\text{DisCount}(\text{add}(I, c, J)) & = \text{DisCount}(I) + \text{DisCount}(J) \\
\text{DisCount}(I \lor J) & = \text{DisCount}(I) + \text{DisCount}(J) + 1 \\
\text{DisCount}(I \land J) & = \text{DisCount}(I) + \text{DisCount}(J) \\
\text{DisCount}(\neg I) & = \text{ConCount}(I)
\end{align*}
\]

*Function ConCount* is defined similarly. Furthermore, we say that $I$ is free of disjunctions, denoted by $\text{DisFree}(I)$, iff $\text{DisCount}(I) = 0$. Similarly, a BIE $I$ is free of conjunctions, denoted by $\text{ConFree}(I)$, iff $\text{ConCount}(I) = 0$.

Note that atomic BIEs coincide with BIEs that are free of disjunctions and free of conjunctions. It will be convenient to describe the components that build a BIE. We will use the notation $\text{HasComp}(I, J)$ to denote that $J$ is a component of $I$. This is captured in the following definition.

**Definition 5.5**

\[
\begin{align*}
\text{HasComp}(I, I) & \\
\text{HasComp}(\text{add}(I, c, J), K) & \iff I = K \lor J = K \\
& \lor \text{HasComp}(I, K) \\
& \lor \text{HasComp}(J, K)
\end{align*}
\]

The components of disjunctions, conjunctions, and negations can be defined similarly.

Dealing with negations during zipping involves recursively zipping the subexpression followed by distributing the negation and restoring the DNF structure by a call to $\text{NotProd}$. Since $\text{NotProd}$ hinges on $\text{OrProd}$, we first state a DNF preservation lemma for $\text{OrProd}$.

**Lemma 5.3** If $I$ and $J$ are in DNF, then $\text{OrProd}(I, J)$ is in DNF.

*Proof:* Suppose $I$ and $J$ are in DNF, say $I = \bigvee_{1 \leq i \leq k} I_i$ and $J = \bigvee_{1 \leq j \leq l} J_j$ such that all subexpressions are atomic. Then, $\text{OrProd}(I, J) = \bigvee_{1 \leq i \leq k, 1 \leq j \leq l} I_i \land J_j$.

The generalised version $\text{OrProd}'$ can be expressed in terms of its original dyadic version. This is used in the lemma below to show that $\text{NotProd}$ does not affect the DNF property.

**Lemma 5.4** If $I$ is in DNF, then $\text{NotProd}(I)$ is in DNF.

*Proof:* Let $I$ be in DNF. By definition, $\text{NotProd}(I)$ can be written in terms of the original $\text{OrProd}$. Thus, by repeated application of lemma 5.3, $\text{NotProd}(\text{Zip}(I))$ is in DNF.
If a BIE does not contain dyadic operators, zipping it does not alter it. This means that for atomic BIEs, zipping instantiates the identity function. As a special case, index expressions are not modified by zipping.

**Lemma 5.5** Atomic(I) \( \Rightarrow \) Zip(I) = I

We now prove an important result for our investigation into expansion. The lemma below consists of two parts corresponding to the two phases of zipping. The first part states that Zip\(\text{Or}\) promotes all disjunctions. Furthermore, it shows that within negated components, all dyadic operators are zipped. The second part states that applying Zip\(\text{And}\) after Zip\(\text{Or}\) yields a BIE in DNF. The corollary to the lemma states the result more tersely: a zipped BIE is in DNF.

**Theorem 5.1 (Zip\(\text{Or}\))** For any BIE I, Zip\(\text{Or}(I) = \bigvee_{1 \leq i \leq k} I_i\), for some \(k \geq 1\), where

1. for all \(1 \leq i \leq k\) : DisFree(I\(_i\)), and
2. HasComp(I, \(-I\)) \(\Rightarrow\) Atomic(\(-I\)).

**Proof:** The proof is done by structural induction on BIEs. The basis for the induction is provided by single terms. The induction step consists of the remaining four forms of BIEs.

**Term** For any term \(t \in T\), Zip\((t) = t\), which trivially is in DNF.

The proofs of the four cases of the induction step, as provided below, consist of two parts: a proof concerning the outcome of Zip\(\text{Or}\) and one concerning Zip\(\text{And}\).

**Add** Suppose the BIE under consideration is of the form add(I, c, J).

**Zip\(\text{Or}\)** By IH,

\[
\text{Zip\(\text{Or}(I) = \bigvee_{1 \leq i \leq k_1} I_i\) and Zip\(\text{Or}(J) = \bigvee_{1 \leq j \leq k_2} J_j\)}
\]

for some \(k_1\) and \(k_2\). In addition, all \(I_i\) and \(J_j\) are of the desired form. That is, they do not contain disjunctions and every negated component is atomic. The result of Zip\(\text{Or}(\text{add}(I, c, J))\) is

\[
\text{OrCons(Zip\(\text{Or}(I), c, Zip\(\text{Or}(J)) = \bigvee_{1 \leq i \leq k_1, 1 \leq j \leq k_2} \text{add}(I_i, c, J_j)}
\]

where rebuilding the composed structures does not corrupt the desired form.

**Zip\(\text{And}\)** Left to the reader.
5.4. ZIPPING OPERATORS IN BIES

**Dis** Suppose the BIE under consideration is of the form $I \lor J$.

**ZipOr** Left to the reader.

**ZipAnd** By IH, the zipped subexpressions are in DNF, say

$$ZipAnd(I) = \bigvee_{1 \leq i_1 \leq k_1, 1 \leq j_1 \leq l_1} I_{i_1,j_1} \quad \text{and} \quad ZipAnd(J) = \bigvee_{1 \leq i_2 \leq k_2, 1 \leq j_2 \leq l_2} I_{i_2,j_2}$$

Clearly, the result $ZipAnd(I) \lor ZipAnd(J)$ also is in DNF.

**Con** Left to the reader.

**Neg** Suppose the BIE under consideration is of the form $\neg I$.

**ZipOr** By combining both parts of the IH, $Zip(I)$ is in DNF. Then, by lemma 5.4, $NotProd(Zip(I))$ is also in DNF.

**ZipAnd** By IH, $\neg I$ is atomic since it forms a negated component. By lemma 5.5, $Zip(\neg I) = \neg I$, which trivially is in DNF.

**Corollary 5.1** For any BIE $I$, $Zip(I)$ is in DNF.

Idempotency of the zip operator is stated in the next lemma. It is used in some of the proofs in the remainder of this chapter.

**Lemma 5.6** $Zip(Zip(I)) = Zip(I)$

BIEs in disjunctive normal form can be matched by slightly altered similarity functions for index expressions ([WBW98c]). The propositional part is dealt with by a translation to a mathematical sum of products and the atomic part by a similarity function for index expressions augmented with negations. The normalised structure after zipping thus eases the design of operations on BIEs.

5.4.5 Dyadic Congruence and Equivalence

Syntactical variety in BIEs is reduced by zipping. In particular, all nested dyadic operators are unnested yielding an equivalent BIE in DNF. Equivalence of BIEs should normalise the order of the arguments of the logical operators. To this end, the notion of dyadic congruence is introduced, normalising reflexivity, commutativity, and associativity of the dyadic operators.
Definition 5.6 Dyadic congruence of BIEs, represented by \( \sim \), is defined as

\[
\begin{align*}
I & \sim I \\
I \lor J & \sim J \lor I \\
I \land J & \sim J \land I \\
(I \lor J) \lor K & \sim I \lor (J \lor K) \\
(I \land J) \land K & \sim I \land (J \land K)
\end{align*}
\]

The following lemma states some elementary properties of dyadic congruence. It implies dyadic congruence is an equivalence order on BIEs.

Lemma 5.7 \( \sim \) is reflexive, symmetric, and transitive.

Dyadic congruence is properly general and powerful for the purpose of examining expansion. It should be noted, however, that different choices could be made. Although these might be highly intuitive, they unnecessarily complicate the examination. The following example illustrates some intuitive congruences that are not included in our definition.

Example 5.4 Seemingly standard congruences like \( I \sim I \land I \) and \( I \sim I \lor I \) are not incorporated in our definition. Although, from a logical point of view, this congruence is elementary, it diverts from examining expansion. The reason for this is that this congruence spoils the property that congruent BIEs have the same dyadic count. Since expansion is defined in terms of dyadic count, this complicates matters. In addition, differences in order of subexpressions are not normalised by \( \sim \). For instance, the BIEs conference on IT in Holland and conference in Holland on IT do not fall under the definition. Expansion focuses on the propositional structure of BIEs, not on the structure of their atomic part. Therefore, after zipping, dyadic congruence only needs to take logical similarities into account. DeMorgan's laws do not have to be modeled by congruence, since negations are zipped downward.

As explained before, normal form transformation, as described in section 5.4.3, forms the basis for our notion of semantical equivalence of BIEs. Two BIEs are equivalent, denoted \( I \equiv J \), iff their zipped forms are congruent. That is, their logical structure may be different as long as their content is the same.

Definition 5.7 \( I \equiv J \iff \text{Zip}(I) \sim \text{Zip}(J) \)

Lemma 5.8 \( \equiv \) is reflexive, symmetric, and transitive.

Zipping a BIE results in a semantically equivalent one.

Lemma 5.9 \( I \equiv \text{Zip}(I) \)
The following important lemma, addressing equivalence under substitution, is used later in this chapter. The substitution lemma states that equivalence is preserved under substitution of equivalent subexpressions. Symmetric results are not explicitly included in the substitution lemma.

**Lemma 5.10 Substitution lemma**

\[
I \equiv J \Rightarrow I \lor K \equiv J \lor K \\
I \land K \equiv J \land K \\
\text{add}(I, c, K) \equiv \text{add}(J, c, K) \\
\neg I \equiv \neg J
\]

The proof of the substitution lemma requires some additional theory, which is provided first. The proof is done by making the two phases in zipping, i.e. ZipOr and ZipAnd, explicit. This is captured in definition 5.8 by the notion of phased congruence. This allows for a translation into well-understood permutations. Preservation of equivalence is examined for all cases of zipping. It turns out that disjunctions follow a simple distribution scheme (lemma 5.13). However, the other cases are more complex and require more elaborate investigation. Lemma 5.14 provides the required insights for these cases. Finally, the proof of the substitution lemma itself is given by integrating these steps. The details of the actual proof comprise the remainder of this subsection, followed by an examination of expansion in section 5.5.

Zipping yields BIEs in DNF, which consist of a three-layered structure: a disjunction of conjunctions of atomic BIEs. This means that for zipped BIEs, dyadic congruence equals permutation.

**Lemma 5.11 Dyadic congruence equals permutation** For zipped BIEs, there exists a permutation \( \pi : 1..k \rightarrow 1..k \) such that

\[
\bigvee_{1 \leq i \leq k} I_i \sim \bigvee_{1 \leq j \leq k} J_j \iff I_i \sim J_{\pi(i)} \\
\bigwedge_{1 \leq i \leq k} I_i \sim \bigwedge_{1 \leq j \leq k} J_j \iff I_i = J_{\pi(i)}
\]

Phased congruence, as defined below, explicitly states an intermediate result of two-phased zipping.

**Definition 5.8 Phased Congruence** \( I \sim \lor J \) iff \( I = \bigvee_{1 \leq i \leq k} I_i \) and \( J = \bigvee_{1 \leq j \leq k} J_j \) such that DisFree\((I_i)\) and DisFree\((J_j)\) and there is a permutation \( \pi \) such that ZipAnd\((I_i) \sim \) ZipAnd\((J_{\pi(i)})\).
Dyadic congruence after zipping coincides with phased congruence after zipping disjunctions only.

**Lemma 5.12** Explicit two-stage zip

\[ \text{Zip}(I) \sim \text{Zip}(J) \iff \text{ZipOr}(I) \sim \text{ZipOr}(J) \]

**Proof:**

\[ \text{Zip}(I) \sim \text{Zip}(J) \]

*By definition of Zip,*

\[ \text{ZipAnd}(\text{ZipOr}(I)) \sim \text{ZipAnd}(\text{ZipOr}(J)) \]

*By the general form after zipping (disjunctions),*

\[ \text{ZipAnd}(\bigvee I_i) \sim \text{ZipAnd}(\bigvee J_j) \]

*By distribution of ZipAnd over disjunctions,*

\[ \bigvee \text{ZipAnd}(I_i) \sim \bigvee \text{ZipAnd}(J_j) \]

*By lemma 5.11,*

\[ \exists \pi : \text{ZipAnd}(I_i) \sim \text{ZipAnd}(J_{\pi(i)}) \]

*By definition 5.8,*

\[ \text{ZipOr}(I) \sim \text{ZipOr}(J) \]

The following lemma illustrates the distribution of zipping over a disjunction.

**Lemma 5.13**

\[ \text{Zip}(I \lor J) = \text{Zip}(I) \lor \text{Zip}(J) \]

**Proof:**

\[ \text{ZipAnd}(\text{ZipOr}(I \lor J)) = \text{ZipAnd}(\text{ZipOr}(I) \lor \text{ZipOr}(J)) \]

\[ = \text{ZipAnd}(\text{ZipOr}(I)) \lor \text{ZipAnd}(\text{ZipOr}(J)) \]

\[ = \text{Zip}(I) \lor \text{Zip}(J) \]

The next lemma provides insights in the preservation of dyadic congruence during zipping for the remaining cases.
Lemma 5.14 Preservation of dyadic congruence

\textbf{AndCons} \quad \text{Let } \text{DisFree}(I), \text{DisFree}(J), \text{and } \text{DisFree}(K). \text{ Then, }
\begin{align*}
\text{ZipAnd}(I) \sim \text{ZipAnd}(J) & \Rightarrow \text{ZipAnd}(\text{add}(I, c, K)) \sim \text{ZipAnd}(\text{add}(J, c, K)) \\
\text{OrCons} & \\
\text{ZipOr}(I) \sim \text{ZipOr}(J) & \Rightarrow \text{ZipOr}(\text{add}(I, c, K)) \sim \text{ZipOr}(\text{add}(J, c, K)) \\
\text{OrProd} & \\
\text{ZipOr}(I) \sim \text{ZipOr}(J) & \Rightarrow \text{ZipOr}(I \land K) \sim \text{ZipOr}(J \land K) \\
\text{OrProd’} & \quad \text{Let } I_1, \ldots, I_n \text{ and } J_1, \ldots, J_n \text{ be disjunctions of atomic BIEs. Then }
\exists \pi : I_i \sim J_{\pi(i)} \Rightarrow \text{OrProd’}(I_1, \ldots, I_n) \sim \text{OrProd’}(J_1, \ldots, J_n) \\
\text{NotProd} & \\
\text{Zip}(I) \sim \text{Zip}(J) & \Rightarrow \text{NotProd}(\text{Zip}(I)) \sim \text{NotProd}(\text{Zip}(J))
\end{align*}

\textbf{Proof:}

\textbf{AndCons} \quad \text{Assume } \text{ZipAnd}(I) \sim \text{ZipAnd}(J) \text{ and } \text{DisFree}(I), \text{DisFree}(J), \text{and } \text{DisFree}(K) \text{ for some } K. \text{ Assume, without loss of generality, } \text{ZipAnd}(I) = \bigwedge I_i, \text{ZipAnd}(J) = \bigwedge J_j, \text{ and } \text{ZipAnd}(K) = \bigwedge K_k \text{ such that all } I_i, J_j, \text{ and } K_k \text{ are atomic. The congruence can thus be restated as }
\begin{align*}
\bigwedge I_i & \sim \bigwedge J_j \\
\text{By lemma 5.11, there is some permutation } \pi \text{ such that }
I_i = J_{\pi(i)}
\end{align*}

\text{Composing equal atomic subexpressions with an auxiliary atomic subexpression preserves equality (and atomicity)}
\begin{align*}
\text{add}(I_i, c, K_k) & = \text{add}(J_{\pi(i)}, c, K_k) \\
\text{Again with lemma 5.11, }
\begin{align*}
\bigwedge \text{add}(I_i, c, K_k) & \sim \bigwedge \text{add}(J_{\pi(i)}, c, K_k) \\
\text{which, by definition of AndCons leads to }
\text{AndCons}(\text{ZipAnd}(I), c, \text{ZipAnd}(K)) & \sim \text{AndCons}(\text{ZipAnd}(J), c, \text{ZipAnd}(K)) \\
\text{which, by definition of ZipAnd equals }
\text{ZipAnd}(\text{add}(I, c, K)) & \sim \text{ZipAnd}(\text{add}(J, c, K))
\end{align*}
\end{align*}
**OrCons** Assume $\text{ZipOr}(I) \sim \lor \text{ZipOr}(J)$. By definition 5.8, $I = \lor I_i$ and $J = \lor J_j$ such that $\text{DisFree}(I_i)$, $\text{DisFree}(J_j)$ and for some permutation $\pi$

$$\text{ZipAnd}(I_i) \sim \text{ZipAnd}(J_{\pi(i)})$$

Without loss of generality, assume $\text{ZipOr}(K) = \lor K_k$ such that $\text{DisFree}(K_k)$. By the previous case of this lemma, for all congruent pairs $\text{ZipAnd}(I_i)$ and $\text{ZipAnd}(J_{\pi(i)})$,

$$\text{ZipAnd}(\text{add}(I_i, c, K_k)) \sim \text{ZipAnd}(\text{add}(J_{\pi(i)}, c, K_k))$$

According to the definition of phased congruence (def. 5.8), this equals

$$\lor_{i,k} \text{add}(I_i, c, K_k) \sim \lor_{j,k} \text{add}(J_j, c, K_k)$$

which, by definition of OrCons yields

$$\text{OrCons}(\lor_{i,k} I_i, c, K_k) \sim \lor_{j,k} \text{OrCons}(J_j, c, K_k)$$

which, by definition of ZipOr gives

$$\text{ZipOr}(\text{add}(I_i, c, K_k)) \sim \lor \text{ZipOr}(\text{add}(J_j, c, K_k))$$

**OrProd** From the assumption $\text{ZipOr}(I) \sim \lor \text{ZipOr}(J)$ we obtain $\lor I_i \sim \lor J_j$ and thus $\exists \pi : \text{ZipAnd}(I_i) \sim \text{ZipAnd}(J_{\pi(i)})$. Without loss of generality, assume $\text{ZipOr}(K) = \lor K_k$, such that $\text{DisFree}(K_k)$. In particular, we may augment the congruences to obtain

$$\text{ZipAnd}(I_i) \land \text{ZipAnd}(K_k) \sim \text{ZipAnd}(J_{\pi(i)}) \land \text{ZipAnd}(K_k)$$

From this, we obtain by definition of ZipAnd,

$$\text{ZipAnd}(I_i \land K_k) \sim \text{ZipAnd}(J_{\pi(i)} \land K_k)$$

and by definition 5.8

$$\lor_{i,k} I_i \land K_k \sim \lor_{j,k} J_j \land K_k$$

which, by definition of OrProd means

$$\text{OrProd}(\text{ZipOr}(I), \text{ZipOr}(K)) \sim \lor \text{OrProd}(\text{ZipOr}(J), \text{ZipOr}(K))$$

which, by definition of ZipOr yields

$$\text{ZipOr}(I \land K) \sim \lor \text{ZipOr}(J \land K)$$
5.4. ZIPPING OPERATORS IN BIES

**OrProd'** Assume $I_i = I_{i1} \lor \cdots \lor I_{in}$ such that all subexpressions are atomic. Furthermore, assume the same holds for all $J_j$. In addition, assume $\exists \pi : I_i \sim J_{\pi(i)}$. This means there exists a $\pi^2 : N^2 \rightarrow N^2$ such that

$$I_{i,j} \sim J_{\pi^2(i,j)}$$

Furthermore, the result of OrProd' is

$$\text{OrProd}'(I_1, \ldots, I_n) = \bigvee I_{1,i1} \land \cdots \land I_{n,in}$$

That is, the result is a disjunction of all possible conjunctions in which one atomic subexpression per $I_i$ is included. This means that for all such conjunctions

$$I_{1,i1} \land \cdots \land I_{n,in} \sim J_{\pi^2(1,i1)} \land \cdots \land J_{\pi^2(n,in)}$$

This allows lemma 5.11 to conclude

$$\bigvee I_{1,i1} \land \cdots \land I_{n,in} \sim \bigvee J_{\pi^2(1,i1)} \land \cdots \land J_{\pi^2(n,in)}$$

which, by definition of OrProd' results in

$$\text{OrProd}'(I_1, \ldots, I_n) \sim \text{OrProd}'(J_1, \ldots, J_n)$$

**NotProd (sketch)** Assume that Zip($I$) $\sim$ Zip($J$). Without loss of generality, assume Zip($I$) = $\bigvee I_{i,j}$ and Zip($J$) = $\bigvee J_{k,l}$. As a consequence,

$$\exists \pi : I_i \sim J_{\pi(i)}$$

where $I_i$ is of the form $I_{i1} \land \cdots \land I_{in}$. The intermediate forms $\bigvee \neg I_{i,j}$ and $\bigvee \neg J_{k,l}$, obtained by distributing the negation over the propositional structures of Zip($I$) and Zip($J$), are congruent as well since the only effects of the distribution are inverting of dyadic operators and negating atomic subexpressions. Note that, when $I$ and $J$ are atomic, so are $\neg I$ and $\neg J$ and in that case $I \sim J$ implies $\neg I \sim \neg J$.

In the intermediate form, the permutation still serves as basis for congruence: $I_i \sim J_{\pi(i)}$. (Only, $I_i$ is now of the form $\neg I_{1,i1} \lor \cdots \lor \neg I_{n,in}$.) Therefore, according to the previous case of this lemma

$$\text{OrProd}'(I_1, \ldots, I_n) \sim \text{OrProd}'(J_1, \ldots, J_n)$$

which, by definition of NotProd results in

$$\text{NotProd}(\text{Zip}(I)) \sim \text{NotProd}(\text{Zip}(J))$$
Making avail of the abovementioned theory, the substitution lemma can be concretely proven.

**Proof of lemma 5.10:**
Assume \( I \equiv J \). This means \( \text{Zip}(I) \sim \text{Zip}(J) \).

**Terms Trivial.**

**Add**  
To show: \( \text{add}(I, c, K) \equiv \text{add}(J, c, K) \), or, \( \text{Zip}([\text{add}(I, c, K)]) \sim \text{Zip}([\text{add}(J, c, K)]) \).

**ZipOr**  
From \( \text{Zip}(I) \sim \text{Zip}(J) \), by lemma 5.12, \( \text{ZipOr}(I) \sim \lor \text{ZipOr}(J) \), which allows lemma 5.14 to conclude

\[
\text{ZipOr}([\text{add}(I, c, K)]) \sim \lor \text{ZipOr}([\text{add}(J, c, K)])
\]

Then, by lemma 5.12,

\[
\text{Zip}([\text{add}(I, c, K)]) \sim \text{Zip}([\text{add}(J, c, K)])
\]

**ZipAnd**  
Since \( I \) and \( J \) are the result of \( \text{ZipOr} \), which precedes \( \text{ZipAnd} \), \( \text{DisFree}(I) \) and \( \text{DisFree}(J) \) and all their negated subexpressions contain no dyadic operators. Combining this with the assumption leads to \( \text{ZipAnd}(I) \sim \text{ZipAnd}(J) \). Then, by lemma 5.14.5, for \( K \) such that \( \text{DisFree}(K) \),

\[
\text{ZipAnd}([\text{add}(I, c, K)]) \sim \text{ZipAnd}([\text{add}(J, c, K)])
\]

And, by the mentioned properties of \( I \) and \( J \), this means

\[
\text{Zip}([\text{add}(I, c, K)]) \sim \text{Zip}([\text{add}(J, c, K)])
\]

**Dis**  
The assumption \( \text{Zip}(I) \sim \text{Zip}(J) \) combined with the trivial result \( \text{Zip}(K) \sim \text{Zip}(K) \), yields

\[
\text{Zip}(I) \lor \text{Zip}(K) \sim \text{Zip}(J) \lor \text{Zip}(K)
\]

By lemma 5.13, this results in

\[
\text{Zip}(I \lor K) \sim \text{Zip}(J \lor K)
\]

which, by definition 5.7 gives

\[
I \lor K \equiv J \lor K
\]

**Con**  
To show: \( I \land K \equiv J \land K \), or, \( \text{Zip}(I \land K) \sim \text{Zip}(J \land K) \).
5.5 Expansion by Zipping Operators

Zipping a BIE results in an equivalent BIE in disjunctive normal form. Generally, the zipped BIE requires more operators than the original one. This is called expansion. In section 5.5.1, we identify a measure for expansion, illustrate that differences in expansion may occur, and provide some initial results on the extremes of expansion. Section 5.5.2 compares our measure to other measures for expansion.

5.5.1 Dyadic Count as Measure for Expansion

Zipping may increase the size of BIEs, which was illustrated by example 5.3. The increase in size of BIEs caused by zipping is called expansion. The number of disjunctions and conjunctions, the so-called dyadic count, serves as a measure for the size of BIEs. Since a key feature of expansion is an increase in the number of dyadic operators (dis- and conjunctions), the dyadic count can also serve as a basis for defining expansion.
Definition 5.9 The expression \([I]\) denotes the dyadic count of \(I\) which is inductively defined as

\[
\begin{align*}
\text{Term} & \quad [t] &= 0 \\
\text{Add} & \quad [\text{add}(I, c, J)] &= [I] + [J] \\
\text{Dis} & \quad [I \lor J] &= 1 + [I] + [J] \\
\text{Con} & \quad [I \land J] &= 1 + [I] + [J] \\
\text{Neg} & \quad \lnot[I] &= [I]
\end{align*}
\]

Congruent BIEs have the same dyadic count.

Lemma 5.15 \(I \sim J \Rightarrow [I] = [J]\)

The observation that zipping does not decrease the dyadic count leads to the following lemma. Since no dyadic operator is discarded of, the dyadic count in the zipped result must be at least the dyadic count of the original.

Lemma 5.16 \([I] \leq [\text{Zip}(I)]\)

Lemma 5.16 can be proven by inspection of the code for zipping. Doing this, one finds that for every dyadic operator on the left hand side of an equation there also occurs (at least) one on the right hand side. This means that zipping does not decrease the number of dyadic operators. In section 5.6.2, it will be shown that this lower bound is tight.

The following three lemma’s present when the dyadic count of a BIE increases by zipping. In turn, composed, conjunctive, and negated BIEs are treated.

Lemma 5.17 \(\lnot(\text{Atomic}(I_1) \lor \text{Atomic}(I_2)) \Rightarrow [\text{Zip}(\text{add}(I_1, c, I_2))] > [\text{add}(I_1, c, I_2)]\)

Lemma 5.18 \(\lnot(\text{DisFree}(I_1) \lor \text{DisFree}(I_2)) \Rightarrow [\text{Zip}(I_1 \land I_2)] > [I_1 \land I_2]\)

Lemma 5.19 \(\lnot(\text{DisFree}(I) \lor \text{ConFree}(I)) \Rightarrow [\text{Zip}(\lnot I)] > [I]\)

The lemma identifies exactly when BIEs cause an increase in dyadic count. The lemma is used in section 5.6.6 in deriving a lower bound on expansion of BIEs.

Lemma 5.20 Let \(I\) be in DNF.

\[
\text{DisFree}(I) \lor \text{ConFree}(I) \Leftrightarrow [\text{NotProd}(I)] = [I]
\]

Proof: We show the implication to the right, leaving the other part for the reader.
The next lemma states that zipped equivalent BIEs have the same dyadic count. It is used in proofs later in this chapter.

**Lemma 5.21** \( I = J \implies \lvert \text{Zip}(I) \rvert = \lvert \text{Zip}(J) \rvert \)

**Proof:** Consequence of definition 5.7 and lemma 5.15.

We now introduce expansion of a BIE as the ratio between the dyadic counts of its zipped and its original versions. Expansion thus is a measure for the increase in dyadic count effectuated by zipping.

**Definition 5.10** The expansion of a BIE \( I \) is defined as

\[
\text{Exp}(I) = \frac{\lvert \text{Zip}(I) \rvert}{\lvert I \rvert}
\]

for \( \lvert I \rvert \neq 0 \). For \( \lvert I \rvert = 0 \), the expansion is 1.

Expansion has a lower and an upper bound. With lemma 5.16, a lower bound of 1 on expansion is obtained, evoked by BIEs whose dyadic count remains unchanged by zipping. For example, lemma 5.5 suggests this is the case for atomic BIEs. However, non-atomic BIEs may also support the lower bound on expansion. The next two sections of this chapter focus on minimally and maximally expanding BIEs, giving explicit characterisations, tightness results on the bounds, and quantitative experiments.

In between the lower and upper bound, intermediate values for expansions can be obtained. The expansion of BIEs depends on the spreading of the Boolean operators. This is illustrated in the next example.
Example 5.5 Consider zipping disjunctions in the case of a composed BIE \( \text{add}(I, c, J) \). For reasons of clarity, assume ZipOr\((I)\) and ZipOr\((J)\) do not contain conjunctions. Suppose that ZipOr\((I)\) and ZipOr\((J)\) contain a total of \( k \) disjunctions, \( k_1 \) of which reside in ZipOr\((I)\):

\[
\text{ZipOr}(I) = \bigvee_{1 \leq i \leq k_1} I_i \text{ and } \text{ZipOr}(J) = \bigvee_{1 \leq j \leq k - k_1} J_j
\]

The way the disjunctions are spread over ZipOr\((I)\) and ZipOr\((J)\), as given by the values of \( k_1 \) and \( k - k_1 \), influences the dyadic count of the result

\[
\text{OrCons}(\text{ZipOr}(I), c, \text{ZipOr}(J)) = \bigvee_{1 \leq i \leq k_1, 1 \leq j \leq k - k_1} \text{add}(I_i, c, J_j)
\]

The number of disjunctions in the result is \((k_1 + 1)(k - k_1 + 1) - 1 = kk_1 - k_1^2 + k\). The expansion is thus \( k_1 - \frac{k_1^2}{k} + 1 \). This function is plotted in Figure 5.9.

![Figure 5.9: Differences in expansion for OrCons.](image)

In Figure 5.9, we see that the expansion is positive. Furthermore, we see that for \( k_1 = 0 \) or \( k_1 = k \), expansion is minimal. That is, if the disjunctions all reside in one of the components, expansion is minimal. The maximal expansion, resulting for \( k/2 \) (for even \( k \)), is \( 1 + k/4 \). That is, if the disjunctions are equally divided over both components, then expansion is maximal. For odd \( k \), both \( \lfloor k/2 \rfloor \) and \( \lceil k/2 \rceil \) give rise to maximal expansion.

As example 5.5 showed, differences in expansion may be caused by the form of BIEs. Even for equivalent BIEs of different form, differences in expansion may occur. Minimally and maximally expanding BIEs are defined in terms of their expansion in the next definition. Minimally (maximally) expanding BIEs achieve the smallest (largest) possible expansion in their equivalence class.
5.5. EXPANSION BY ZIPPING OPERATORS

Definition 5.11 Minimally and maximally expanding BIEs.

\[
\text{min-exp}(I) \iff \forall J[I \equiv J \Rightarrow \text{Exp}(I) \leq \text{Exp}(J)]
\]

\[
\text{max-exp}(I) \iff \forall J[I \equiv J \Rightarrow \text{Exp}(I) \geq \text{Exp}(J)]
\]

Minimally expanding BIEs have minimally expanding subexpressions. In other words, minimal expansion is a property that is recursively preserved over the structure of BIEs.

Lemma 5.22

\[
\text{min-exp}(\text{add}(I_1, c, I_2)) \Rightarrow \text{min-exp}(I_1) \text{ and } \text{min-exp}(I_2)
\]

\[
\text{min-exp}(I_1 \lor I_2) \Rightarrow \text{min-exp}(I_1) \text{ and } \text{min-exp}(I_2)
\]

\[
\text{min-exp}(I_1 \land I_2) \Rightarrow \text{min-exp}(I_1) \text{ and } \text{min-exp}(I_2)
\]

\[
\text{min-exp}(\neg I) \Rightarrow \text{min-exp}(I)
\]

Proof: We only show the second case; the remaining cases follow a similar line of reasoning. So, suppose \(\text{min-exp}(I_1 \lor I_2)\). We show that the leftmost subexpression is minimally expanding, i.e. \(\text{min-exp}(I_1)\). A similar line of reasoning holds for the rightmost subexpression. From \(\text{min-exp}(I_1 \lor I_2)\), we obtain by definition 5.11

\[
\forall J[I_1 \lor I_2 \equiv J \Rightarrow \text{Exp}(I_1 \lor I_2) \leq \text{Exp}(J)]
\]

This implies

\[
\forall J[I_1 \equiv J \Rightarrow \text{Exp}(I_1 \lor I_2) \leq \text{Exp}(J \lor I_2)]
\]

From \(I_1 \equiv J\), lemma 5.10 yields \(I_1 \lor I_2 \equiv J \lor I_2\). By lemma 5.21, this means \([\text{Zip}(I_1 \lor I_2)] = [\text{Zip}(J \lor I_2)]\). Thus, for all \(J \equiv I_1\),

\[
\frac{[\text{Zip}(I_1 \lor I_2)]}{[I_1 \lor I_2]} \leq \frac{[\text{Zip}(J \lor I_2)]}{[J \lor I_2]}
\]

Combining this with \([\text{Zip}(I_1 \lor I_2)] = [\text{Zip}(J \lor I_2)]\) means \([I_1 \lor I_2] \geq [J \lor I_2]\). Using the definition of dyadic count (def. 5.9), this leads to \([I_1] \geq [J]\). Therefore,

\[
\frac{[\text{Zip}(I_1)]}{[I_1]} \leq \frac{[\text{Zip}(I_1)]}{[J]} \leq \frac{[\text{Zip}(J)]}{[J]}
\]

meaning \(\text{min-exp}(I_1)\).

Similar properties are obtained for maximally expanding BIEs.
**Lemma 5.23**

\[
\begin{align*}
\text{max-exp}(\text{add}(I_1, c, I_2)) & \Rightarrow \text{max-exp}(I_1) \text{ and max-exp}(I_2) \\
\text{max-exp}(I_1 \lor I_2) & \Rightarrow \text{max-exp}(I_1) \text{ and max-exp}(I_2) \\
\text{max-exp}(I_1 \land I_2) & \Rightarrow \text{max-exp}(I_1) \text{ and max-exp}(I_2)
\end{align*}
\]

**Proof:** We only prove the first case, the remaining cases follow a similar line of argument. Assume \(\text{max-exp}(\text{add}(I_1, c, I_2))\). We show \(\text{max-exp}(I_1)\), leaving the result for \(I_2\) to the reader.

\[
\forall J [\text{add}(I_1, c, I_2) \equiv J] \Rightarrow \text{Exp}(\text{add}(I_1, c, I_2)) \leq \text{Exp}(J)
\]

This implies

\[
\forall J [I_1 \equiv J] \Rightarrow \text{Exp}(\text{add}(I_1, c, I_2)) \leq \text{Exp}(\text{add}(J, c, I_2))
\]

meaning

\[
\frac{\text{Zip}(\text{add}(I_1, c, I_2))}{\text{add}(I_1, c, I_2)} \leq \frac{\text{Zip}(\text{add}(J, c, I_2))}{\text{add}(J, c, I_2)}
\]

which, by \(\text{Zip}(\text{add}(I_1, c, I_2)) = \text{Zip}(\text{add}(J, c, I_2))\) leads to \([I_1] \leq [J]\), yielding

\[
\text{Exp}(I_1) \geq \text{Exp}(J)
\]

**Example 5.6** There are BIEs that are both minimally and maximally expanding. This holds, for instance, for atomic BIEs. In addition, it holds for every BIE with a single nested dyadic operator.

A compact BIE is defined to contain the smallest possible dyadic count within its equivalence class.

**Definition 5.12 Compactness**

\[
\text{compact}(I) \equiv \forall J[I \equiv J] \Rightarrow [I] \leq [J]
\]

Maximally expanding BIEs coincide with compact BIEs, as shown by the following lemma.

**Lemma 5.24**

\[
\text{max-exp}(I) \Leftrightarrow \text{compact}(I)
\]

**Proof:**

\[
\begin{align*}
\text{max-exp}(I) & \Leftrightarrow \forall J[I \equiv J] \Rightarrow \text{Exp}(I) \geq \text{Exp}(J) \quad (\text{Def. 5.11}) \\
& \Leftrightarrow \forall J[I \equiv J] \Rightarrow \frac{\text{Zip}(I)}{[I]} \geq \frac{\text{Zip}(J)}{[J]} \quad (\text{Def. 5.10}) \\
& \Leftrightarrow \forall J[I \equiv J] \Rightarrow [I] \leq [J] \quad (\text{Lemma 5.21}) \\
& \Leftrightarrow \text{compact}(I) \quad (\text{Def. 5.12})
\end{align*}
\]

All necessary ingredients are now available for a thorough investigation of a lower and upper bound on expansion of BIEs. This will be done in sections 5.6 and 5.7. First, we compare our measure for expansion, defined in terms of the dyadic count, with other possible measures.
5.5.2 Other Measures for Expansion

As the dyadic count grows by zipping, the number of terms and connectors increases as well. This also provides a basis for measuring expansion. One would, however, have to cope with asymmetric distribution of terms and connectors over the atomic constituents of the zipped result. This problem does not occur when looking at the dyadic count.

For zipped BIEs, the dyadic count is directly related to the number of atomic BIEs in the resulting proposition. More specifically, the number of atomic subexpressions in zipped BIEs is the dyadic count plus one. The number of terms or connectors in zipped BIEs can thus be defined in terms of the dyadic count of zipped BIEs and information about the unzipped BIE.

**Lemma 5.25**

\[
[\text{Zip}(I)] + 1 \leq \#\text{Terms}(\text{Zip}(I)) \leq ([\text{Zip}(I)] + 1) \times (\#\text{Terms}(I) - [I])
\]

Lemma 5.25 is illustrated to hold by the following observations. Since each atomic BIE consists of at least a single term, the number of terms of a zipped BIE is bounded below by the number of atomic BIEs in it. This bound is tight for BIEs that consist of a single term. The number of terms is bounded above by the product of the number of atomic BIEs and the maximal number of terms per atomic BIE. Since arguments of disjunctions and conjunctions consist of at least a single term, the atomic BIEs loose at least a term for every such operator. Thus, each atomic BIE consists of at most \#Terms(I) – [I] terms. In other words, the bound is tight if all arguments of dyadic operators are single terms.

For BIEs with at least one connector, the following bounds apply.

**Lemma 5.26**

\[
[\text{Zip}(I)] + 1 \leq \#\text{Conn}(\text{Zip}(I)) \leq ([\text{Zip}(I)] + 1) \times \#\text{Conn}(I)
\]

Each resulting atomic BIE will contain at least one connector. Thus, the minimal number of connectors in the zipped result is at least the number of atomic BIEs. Since each atomic BIE may contain all connectors, the upper bound is the product of the number of atomic BIEs and the original number of connectors.

Although the number of negations in a zipped BIE can be bounded, it does not provide a clear measure for expansion. An increase in dyadic count, for instance, does not always imply an increase in the number of negations. Furthermore, negations with dyadic operators in their subexpression effect an increase in dyadic count, whereas other negations do not. Therefore, the expanding effect of zipping is not clearly captured by (an increase in) the number of negations.

**Lemma 5.27**

\[
\#\text{Negs}(I) \leq \#\text{Negs}(\text{Zip}(I)) \leq ([\text{Zip}(I)] + 1) \times \#\text{Negs}(I)
\]
Zipping may copy all negations of the original BIE in all atomic BIEs of the result. This leads to the upper bound of lemma 5.27. A lower bound cannot be specified in terms of the atomic size of the zipped BIE since not every atomic part has to contain negations. Since no negations are discarded of, a lower bound is given by the original number of negations.

5.6 Minimal Expansion

In the previous section, minimally expanding BIEs were defined as the least expanding descriptors of an equivalence class. In this section, these BIEs are paid a closer look. The next section does the same for maximally expanding BIEs.

Minimally expanding BIEs are investigated in a number of steps. In section 5.6.1, we explicitly characterise the class of so called minimal BIEs. Then, section 5.6.2 provides a tight lower bound on the expansion of BIEs by examining the expansion of minimal BIEs. In addition, it shows that minimal BIEs coincide with minimally expanding BIEs, thus validating our characterisation. As an illustration, we provide an example class of minimal BIEs in section 5.6.3. Finally, we analyse the quantitative behaviour of minimal BIEs in section 5.6.4.

5.6.1 Explicit Characterisation of Minimal BIEs

As stated in the introduction, having an explicit characterisation of minimal BIEs eliminates the need to zip them in deciding on their minimality. This is valuable since zipping may be costly. This property can be exploited in, for instance, user interfaces where BIEs are to be presented to the user. Knowing that a BIE is minimal allows for direct computation of the size of the zipped version. From this size, one can derive the required space in a window to represent the zipped BIE or the amount of memory needed for storing it. Furthermore, explicitly characterising minimal BIEs gives us the possibility to efficiently generate and compute the fraction of all BIEs that are minimal.

**Definition 5.13** A BIE $I$ is called minimal, denoted by $\text{IsMin}(I)$, iff it adheres to the inductive definition below.

\[
\begin{align*}
\text{IsMin}(t) & \iff \text{IsMin}(I) \\
\text{IsMin}(\text{add}(I, c, J)) & \iff \text{IsMin}(I) \text{ and } \text{IsMin}(J) \text{ and } (\text{Atomic}(I) \text{ or } \text{Atomic}(J)) \\
\text{IsMin}(I \lor J) & \iff \text{IsMin}(I) \text{ and } \text{IsMin}(J) \\
\text{IsMin}(I \land J) & \iff \text{IsMin}(I) \text{ and } \text{IsMin}(J) \text{ and } (\text{DisFree}(I) \text{ or } \text{DisFree}(J)) \\
\text{IsMin}(\neg I) & \iff \text{IsMin}(I) \text{ and } (\text{DisFree}(I) \text{ or } \text{ConFree}(I))
\end{align*}
\]

The explicit characterisation of minimal BIEs describes those BIEs for which the dyadic count after zipping is minimal. During zipping, applying certain functions can increase the dyadic count. In order to guarantee a minimal dyadic count, restrictions must be put on the
arguments of functions. Below, we illustrate definition 5.13 by examining the occurrences of these functions in the code for zipping.

**Term** Zipping terms does not involve any additional functions, delivering the unchanged term as result. Clearly, the dyadic count of terms does not increase by zipping. Therefore, terms are minimal.

**Add** In the composed case of ZipOr, function OrCons is called.

\[ \text{ZipOr}(\text{add}(I, c, J)) = \text{OrCons}(\text{ZipOr}(I), c, \text{ZipOr}(J)) \]

At least one of the subexpressions should be free of dyadic operators. Otherwise, OrCons causes an increase in dyadic count. Because of the recursive calls to ZipOr, the subexpressions must be minimal BIEs.

In the composed case of ZipAnd, function AndCons is called.

\[ \text{ZipAnd}(\text{add}(I, c, J)) = \text{AndCons}(\text{ZipAnd}(I), c, \text{ZipAnd}(J)) \]

By similar arguments, this case results in the same requirements as for ZipOr.

**Dis** For the disjunctive case of both ZipOr and ZipAnd, no additional functions are called. The recursive calls to these functions, however, imply that both arguments should be minimal.

**Con** For the conjunctive case of ZipOr, function OrProd is called.

\[ \text{ZipOr}(I \land J) = \text{OrProd}(\text{ZipOr}(I), \text{ZipOr}(J)) \]

No disjunctions should occur in the arguments of the conjunction. Otherwise, OrProd duplicates the conjunction. Furthermore, both arguments should be minimal.

For the conjunctive case of ZipAnd, no additional functions are called. The recursive calls to ZipAnd, however, imply that both arguments must be minimal.

**Neg** For negated BIEs, ZipOr applies NotProd to a BIE in disjunctive normal form. By definition, NotProd involves the generalised OrProd which, in turn, can be expressed as a series of OrProds. As shown by lemma 5.20, this causes an increase in dyadic count unless the BIE contains at most one type of dyadic operator. Furthermore, the recursive call to ZipOr implies that the argument must be minimal.

Function ZipAnd, in the case of negated BIEs, coincides with identity. Therefore, no constraints on minimal BIEs stem from this case.

The following theorem states that every minimally expanding BIE (def. 5.11) is minimal (def. 5.13).
Chapter 5. Boolean Index Expressions

Theorem 5.2 \( \text{min-exp}(I) \Rightarrow \text{IsMin}(I) \)

Proof: The proof is done by structural induction to BIEs.

Terms Trivial.

Dis Suppose \( I \) is minimally expanding and of the form \( I_1 \lor I_2 \). By Lemma 5.22, we obtain \( \text{min-exp}(I_1) \) and \( \text{min-exp}(I_2) \) which, by the induction hypothesis, leads to \( \text{IsMin}(I_1) \) and \( \text{IsMin}(I_2) \). By definition 5.13, this means that \( \text{IsMin}(I_1 \lor I_2) \).

Add Suppose \( I \) is minimally expanding and of the form \( \text{add}(I_1, c, I_2) \). By Lemma 5.22, we obtain \( \text{min-exp}(I_1) \) and \( \text{min-exp}(I_2) \). Remains to be shown that at least one of the subexpressions does not contain dyadic operators. Lemma 5.9 implies \( \text{add}(I_1, c, I_2) = \text{Zip}(\text{add}(I_1, c, I_2)) \) and thus, by minimal expansion of \( \text{add}(I_1, c, I_2) \),

\[
\text{Exp}(\text{add}(I_1, c, I_2)) \leq \text{Exp}(\text{Zip}(\text{add}(I_1, c, I_2)))
\]

Now, for the sake of argument, assume \( \neg(\text{Atomic}(I_1) \lor \text{Atomic}(I_2)) \). By Lemma 5.17, this leads to \( \lfloor \text{Zip}(\text{add}(I_1, c, I_2)) \rfloor > \lfloor \text{add}(I_1, c, I_2) \rfloor \). Therefore, also using Lemma 5.6,

\[
\frac{\lfloor \text{Zip}(\text{add}(I_1, c, I_2)) \rfloor}{\lfloor \text{add}(I_1, c, I_2) \rfloor} > \frac{\lfloor \text{Zip}^2(\text{add}(I_1, c, I_2)) \rfloor}{\lfloor \text{Zip}(\text{add}(I_1, c, I_2)) \rfloor}
\]

or \( \text{Exp}(\text{add}(I_1, c, I_2)) > \text{Exp}(\text{Zip}(\text{add}(I_1, c, I_2))) \) which gives the required contradiction.

Con Similar, using Lemma 5.18.

Neg Similar, using Lemma 5.19.

The reverse of theorem 5.2 also holds, i.e. minimal BIEs are minimally expanding, as shown in the next subsection.

5.6.2 Lower Bound

In this section, we provide a tight lower bound on the expansion of BIEs. In order to show that the lower bound of Lemma 5.16 is tight, we prove that, for minimal BIEs, the dyadic count does not change by zipping. This tightness result allows us to conclude that every minimal BIE is minimally expanding.

The following theorem lies the basis for the tightness result, which is presented as a corollary. The theorem describes the outcome of ZipOr and ZipAnd, when applied to minimal BIEs. Zipping disjunctions in minimal BIEs does not change the dyadic count. Thus, the result of ZipOr forms a disjunction of \( \text{DisCount}(I) + 1 \) elements that together contain the original number of conjunctions. ZipAnd locally zips these elements without altering the number of conjunctions.

Theorem 5.3 If \( I \) is minimal, then
5.6. MINIMAL EXPANSION

(ZipOr)

\[ \text{ZipOr}(I) = \bigvee_{1 \leq i \leq \text{DisCount}(I)+1} I_i \]

with \( \text{ConCount}(I) = \text{ConCount}(\text{ZipOr}(I)) \) and

(ZipAnd) If \( J = \text{ZipOr}(I) \) then

\[ \text{ZipAnd}(J) = \bigvee_{1 \leq i \leq \text{DisCount}(I)+1} \bigwedge_{1 \leq j \leq i+1} J_{i,j} \]

where \( \sum_{i=1}^{\text{DisCount}(I)+1} t_i = \text{ConCount}(I) \).

Proof:
The proof is done by structural induction on BIEs. We only provide the case of composed BIEs, leaving the remaining cases to the reader.

Add Suppose \( I \) is minimal and of the form \( \text{add}(J, c, K) \). By definition 5.13, at least one of the subexpressions is atomic. Assume, without loss of generality, that \( J \) is atomic. This implies \( \text{DisCount}(I) = \text{DisCount}(K) \) and \( \text{ConCount}(I) = \text{ConCount}(K) \). Since \( \text{Atomic}(J) \), lemma 5.5 yields \( \text{Zip}(J) = J \).

ZipOr By IH,

\[ \text{ZipOr}(K) = \bigvee_{1 \leq i \leq \text{DisCount}(I)+1} K_i \]

such that \( \text{ConCount}(K) = \text{ConCount}(\text{ZipOr}(K)) \). Thus, \( \text{ZipOr}(\text{add}(J, c, K)) = \text{OrCons}(J, c, \bigvee_{1 \leq i \leq \text{DisCount}(I)+1} K_i) = \bigvee_{1 \leq i \leq \text{DisCount}(I)+1} \text{add}(J, c, K_i) \)

such that \( \text{ConCount}(\text{add}(J, c, K)) = \text{ConCount}(\text{ZipOr}(\text{add}(J, c, K))) \).

ZipAnd Suppose \( \text{ZipOr}(I) = \text{add}(J, c, K) \), meaning \( \text{DisCount}(I) = 0 \). By IH,

\[ \text{ZipAnd}(J) = \bigwedge_{1 \leq j \leq \text{ConCount}(I)+1} J_j \]

Thus, \( \text{ZipAnd}(\text{add}(I, c, J)) = \text{AndCons}(I, c, \bigwedge_{1 \leq j \leq \text{ConCount}(I)+1} J_j) = \bigwedge_{1 \leq j \leq \text{ConCount}(I)+1} \text{add}(I, c, J_j) \)

which, since \( \text{DisCount}(I) = 0 \), is in the desired form.
As a consequence of this theorem, the following corollary holds, giving insight in the form and size of zipped minimal BIEs.

**Corollary 5.2** If $I$ is minimal, then

$$\text{Zip}(I) = \bigvee_{1 \leq i \leq \text{DisCount}(I) + 1} \bigwedge_{1 \leq j \leq t_i + 1} J_{i,j}$$

such that $\sum_{i=1}^{\text{DisCount}(I) + 1} l_i = \text{ConCount}(I)$.

The following corollary also is a direct consequence of theorem 5.3: the dyadic count does not change when minimal BIEs are zipped.

**Corollary 5.3** $\text{IsMin}(I) \Rightarrow [\text{Zip}(I)] = [I]$.

By corollary 5.3, the lower bound on expansion from lemma 5.16 is tight: minimal BIEs realise the smallest possible expansion.

**Lemma 5.28** $\text{IsMin}(I) \Rightarrow \text{min-exp}(I)$

**Proof:** Corollary 5.3 implies that minimal BIEs I have $\text{Exp}(I) = 1$. Lemma 5.16 implies this is the minimal expansion value. Thus, all minimal BIEs are minimally expanding.

We may now conclude that minimal BIEs coincide exactly with minimally expanding BIEs. Minimal BIEs thus correctly and fully characterise minimally expanding BIEs: every minimal BIE is minimally expanding and there is no minimally expanding BIE that is not minimal.

**Corollary 5.4** $\text{IsMin}(I) \iff \text{min-exp}(I)$

**Proof:** Direct consequence of theorem 5.2 and lemma 5.28.

### 5.6.3 Example Minimal BIEs

In this subsection, the main results on expansion of minimal BIEs are illustrated by an example. A simple class of minimal BIEs is defined for which a direct computation of the expansion is performed. The example gives a direct computation of the general result on form and size after zipping, as stated in theorem 5.3. In addition, it illustrates the stable dyadic count, as provided in corollary 5.3.
Example 5.7 Minimal Expansion
Simple BIEs containing $k$ nested disjunctions and $l$ nested conjunctions that are minimally expanding are, for instance, of the form $\text{add}(t, c, I)$ where $t$ is a single term and $I$ consists of the $k+l$ operators separated by $k+l+1$ single terms.

For a schematic picture of this form, see Figure 5.10. Furthermore, we require $I$ to be in DNF so that no distributive laws are necessary in zipping disjunctions first. That is,

$$I = \bigvee_{1 \leq i \leq k+1} \bigwedge_{1 \leq j \leq l_i+1} t_{i,j}$$

with the total number of conjunctions being $\sum_{i=1}^{k+1} l_i = l$.

$$\text{ZipOr}(\text{add}(t, c, I)) = \text{OrCons}(\text{ZipOr}(t), c, \text{ZipOr}(I)) = \text{OrCons}(t, c, \bigvee_{1 \leq i \leq k+1} \bigwedge_{1 \leq j \leq l_i+1} t_{i,j}) = \bigvee_{1 \leq i \leq k+1} \text{add}(t, c, \bigwedge_{1 \leq j \leq l_i+1} t_{i,j})$$

$$\text{ZipAnd}(\bigvee_{1 \leq i \leq k+1} \text{add}(t, c, \bigwedge_{1 \leq j \leq l_i+1} t_{i,j})) = \bigvee_{1 \leq i \leq k+1} \text{AndCons}(t, c, \bigwedge_{1 \leq j \leq l_i+1} t_{i,j}) = \bigvee_{1 \leq i \leq k+1} \text{add}(t, c, t_{i,j})$$

This means that $[\text{ZipAnd}(\text{ZipOr}(\text{add}(t, c, I)))] = k + \sum_{i=1}^{k+1} l_i = k + l$.

Note that, although the dyadic count does not increase by zipping the example minimal BIEs, the number of required terms and connectors does. This is a result of unnesting the dyadic operators, which effectuates a loss of subexpression sharing.

5.6.4 Quantitative Analysis of Minimal BIEs
The preceding part of this section provided a mainly qualitative examination of minimal BIEs. This section illustrates the quantitative behaviour of minimal BIEs. In particular, quantitative figures about the fraction of minimal BIEs are required to ascertain their merit. For example, a large fraction of minimal BIEs is valuable for applications that require easy processing of BIEs or a readily derived semantics. On the other hand, applications aiming
at compactly representing complex information needs are not helped by a large fraction of minimal BIEs.

Let $n_{BIE}(t, d, c, n)$ denote the number of possible BIEs with $t$ terms, $d$ disjunctions, $c$ conjunctions, and $n$ negations. Similarly, let $n_{Min}(t, d, c, n)$ denote the number of minimal BIEs with given constituents. Then, the fraction of minimal BIEs can be defined as

$$\text{MinFrac}(t, d, c, n) = \frac{n_{Min}(t, d, c, n)}{n_{BIE}(t, d, c, n)}$$

In order to get representative, generally valid results, the BIEs in the tests should be complex enough to incorporate the full range of syntactical variation. Therefore, BIEs with seven terms and various numbers of logical operators were used.

To distinguish between the effects of the different logical operators, we isolate negations from dyadic operators, thus requiring two separate tests. In both tests, we generated all possible BIEs with the described numbers of constituents and computed the fraction of minimal BIEs. Since minimality of BIEs does not depend on the instantiation of terms and connectors, we abstract from their actual contents. Instead of actual terms and connectors, we used constants representing term and connector positions.

**Negations**

To investigate the influence of negations on the fraction of minimal BIEs, we analysed $\text{MinFrac}(7, 1, 2, n)$ for $1 \leq n \leq 6$. This allows a representative analysis since many different BIEs can be constructed with these constituents. At least seven terms are needed to place three dyadic operators in different subexpressions constructed by the $\text{add}$ operator. In addition, they can be placed in the arguments of other dyadic operators in any possible way. The results are provided in Figure 5.11.

In Figure 5.11, the influence of negations on the fraction of minimal BIEs is depicted. On the x-axis, the number of negations $n$ is plotted, ranging from zero to six. The left y-axis provides percentages. The line through the crosses is to be seen in the light of the left y-axis. A cross $(n, p)$ denotes that $\text{MinFrac}(7, 1, 2, n) = p$. The lines through the dots are to be seen in the light of the right y-axis. Note that this y-axis has a logarithmic scale. A filled circle denotes $(n, n_{BIE}(7, 1, 2, n))$ whereas an empty circle denotes $(n, n_{Min}(7, 1, 2, n))$.

From Figure 5.11, the following conclusions can be drawn. First, the fraction of minimal BIEs is considerable: it decreases from 30 percent for no negations to 25 percent for six negations. The decrease is caused by the constraints for negations: the more negations are placed in a BIE, the smaller the number of possibilities to place all of them in such a way that their subexpressions contain at most one type of dyadic operator.

Second, on a logarithmic scale, the growth of the number of (minimal) BIEs is sub-linear. This illustrates that the factor of growth decreases if the number of negations increases. The reason for this lies in different ways to construct the same BIE. The more negations are added, the larger the number of ways to construct the same BIE.
5.6. MINIMAL EXPANSION

Dyadic Operators

The influence of dyadic operators on the fraction of minimal BIEs is analysed by examining $\text{MinFrac}(7, d, c, 0)$ with varying dyadic count $d + c$. The results are shown in Figure 5.12.

In Figure 5.12, the y-axis denotes percentages. The y-value of a dot labeled by a pair $(d, c)$ represents $\text{MinFrac}(7, d, c, 0)$. Lines are drawn through dots that represent the same dyadic count, i.e. for which $d + c$ is equal. The x-axis denotes the difference in numbers of disjunctions and conjunctions. A dot $(d, c)$ has x-value $d - c$. Thus, the right hand side of the figure, i.e. those dots with a positive x-value, contains dots that represent BIEs with a surplus of disjunctions. We examine dyadic counts up to six, since $\text{nBIE}(t, d, c, n) = 0$ if $d + c \geq t$ and $t = 7$. 
The bowl-form of the lines of Figure 5.12 shows that, for given dyadic count, the fraction of minimal BIEs is directly proportional to the ratio between disjunctions and conjunctions. That is, the fraction of minimal BIEs decreases when the difference between the numbers of disjunctions and conjunctions decreases. Minimal values are obtained for dots on, or just left to, the y-axis. If one type of dyadic operators has the upper hand, the fraction of minimal BIEs is larger.

Not all lines are symmetric in the y-axis: \( \text{MinFrac}(7, d, c, 0) \) is not always equal to its counterpart \( \text{MinFrac}(7, c, d, 0) \). This is illustrated by, for example, dots \((1, 2)\) and \((2, 1)\). For dots with equal distance to the y-axis, those denoting BIEs with a surplus of disjunctions denote higher fractions than those on the left hand side. This means that disjunctions put less restrictions on the form of BIEs in order for them to be minimal than conjunctions. However, BIEs with only one type of dyadic operators are symmetric in this sense: \( \text{MinFrac}(7, d, 0, 0) = \text{MinFrac}(7, 0, d, 0) \). This means that, in absence of one type of dyadic operators, disjunctions and conjunctions put the same restrictions on minimal BIEs. This can be explained from the definition of minimal BIEs: disjunctions put no restrictions on their arguments and the only restriction for conjunctions concerns disjunctions which, then, are absent.

Consider the fraction of minimal BIEs with only disjunctions, as represented by dots \((d, 0)\). With an increase of the number of disjunctions \(d\), the fraction of minimal BIEs first decreases and then increases again. In this context, for composed BIEs to be minimal, at least one of their subexpressions should not contain disjunctions. For small numbers of disjunctions, there are many ways to achieve this since the disjunctions can be placed relatively far apart. For large numbers of disjunctions, few composed BIEs can be constructed by the add operator. Therefore, the number of minimal BIEs is substantial in both (extreme) cases. A similar line of reasoning applies to BIEs with only conjunctions.

In this section, we analysed minimal BIEs. When zipped, minimal BIEs show the smallest possible expansion. This property can be useful since it saves space, e.g. in memory or on a computer screen. The explicit characterisation opened doors for quantitative analyses. In addition, it can be exploited in applications with BIEs in order to prevent or guarantee the construction of minimal BIEs. In the next section, a similar investigation is performed for maximally expanding BIEs.

## 5.7 Maximal Expansion

The previous section provided a tight lower bound on expansion of BIEs. A tight upper bound on expansion is provided in this section by examining so called maximal BIEs.

The examination of maximal BIEs follows a similar pattern as in the previous section. However, negations and dyadic operators are treated separately. The reason for this is that negations cause maximal expansion only if all dyadic operators reside in their subexpression. This means that the expanding effect of negations can be studied separately from that of
dyadic operators. BIEs that do not contain negations are investigated in section 5.7.1. This leads to the introduction of so called maximal positive BIEs. Section 5.7.2 focuses on the expanding effect of negations, exploiting the results from maximal positive BIEs. Section 5.7.3 constitutes an analysis of quantitative behaviour of maximal (positive) BIEs. Finally, section 5.7.4 provides ways to handle maximal BIEs that have impracticle expansion.

### 5.7.1 Maximal Positive BIEs

In this section, maximal expansion for BIEs without negations is investigated in the following steps. An explicit characterisation of maximal positive BIEs is provided first. Then, the expansion of maximal positive BIEs is examined, which leads to an upper bound on expansion for BIEs without negations. Finally, an example class of maximal positive BIEs is provided, so called umbrella BIEs.

#### Explicit Characterisation

Maximal positive BIEs reach maximal expansion for BIE without negations. In other words, there are no other BIEs with the same constituents that reach a larger expansion. The definition of maximal positive BIEs ensures that the dyadic operator are spread nicely. This means that their expanding potential is fully utilised.

**Definition 5.14** A BIE $I$ is called maximal positive, denoted by $\text{IsMaxPos}(I)$, iff it adheres to the inductive definition below.

\[
\begin{align*}
\text{IsMaxPos}(t) & \iff \text{IsMaxPos}(t) \\
\text{IsMaxPos}(\text{add}(I, c, J)) & \iff \text{IsMaxPos}(I) \text{ and } \text{IsMaxPos}(J) \\
\text{IsMaxPos}(I \lor J) & \iff \text{IsE}(I) \text{ and } \text{IsE}(J) \\
\text{IsMaxPos}(I \land J) & \iff \text{IsMaxPos}(I) \text{ and } \text{IsMaxPos}(J) \text{ and } \text{ConFree}(I) \text{ and } \text{ConFree}(J)
\end{align*}
\]

Below, the definition of maximal positive BIEs is explained.

- **Terms** Terms do not contain dyadic operators. Zipping them thus trivially yields a maximal increase in dyadic count.

- **Add** Zipping composed BIE $\text{add}(I, c, J)$ involves

\[
\text{OrCons}(\text{ZipOr}(I), c, \text{ZipOr}(J))
\]

Clearly, if one of the arguments of $\text{ZipOr}$ does not reach a maximal increase in dyadic count, the composed BIE has the same property. Therefore, both subexpressions should be maximal positive. A similar argument applies to $\text{AndCons}$. 
Dis Zipping a disjunction \( I \lor J \) involves

\[
\text{Zip}(I) \lor \text{Zip}(J)
\]

Note that, on the basis of this disjunction, no operator in the subexpressions is duplicated. In other words, the expanding potential of this disjunction is not utilised. The only way in which this does not harm maximal expansion is if there are no nested dyadic operators.

Con Following a similar line of argument as for disjunctions, we conclude from \text{ZipAnd} that the arguments of conjunctions may not contain additional conjunctions. The arguments may, however, contain disjunctions. The reason for this is that \text{ZipOr} precedes \text{ZipAnd}. During \text{ZipOr}, calls to \text{OrProd} copy the conjunctions into both arguments of the resulting disjunction.

Maximally expanding BIEs are defined relative to their equivalence class (see def. 5.11). However, maximal positive BIEs are such that they reach the upper bound of expansion relative to the dyadic operators they contain. Therefore, not every maximally expanding BIE is maximal positive.

**Example 5.8** Consider, for instance, the BIE \( t_1 \lor (t_2 \land t_3) \). This BIE is maximally expanding since, within its equivalence class, there is no more compact representation. However, the example BIE is not maximal positive since the right subexpression of the disjunction is not a classical index expression. With the same dyadic operators, a BIE with larger expansion can be constructed: \( \text{add}(t_1 \lor t_2, c, t_3 \lor t_4) \). This means that the expanding potential of the dyadic operators in the first BIE is not fully utilised.

In the relation between maximally expanding BIEs and maximal positive BIEs, it is thus important to know if a BIE can be represented as a BIE in which the expanding potential of its dyadic operators is fully exploited.

**Definition 5.15** A BIE \( I \) is called representable as maximal positive iff there is a BIE \( J \) such that \( I \equiv J \) and IsMaxPos(\( J \)).

As example 5.8 showed, not every maximally expanding BIE is maximal positive. However, if a maximally expanding BIE is representable as maximal positive, it is maximal positive.

**Lemma 5.29** Let \( I \) be a BIE representable as maximal positive and not containing negations. Then,

\[
\text{max-exp}(I) \Rightarrow \text{IsMaxPos}(I)
\]

**Proof:** Suppose \( I \) is maximally expanding and representable as maximal positive.
5.7. MAXIMAL EXPANSION

Term Trivial.

Add Assume $\text{max-exp}(\text{add}(I_1, c, I_2))$. Lemma 5.23 yields $\text{max-exp}(I_1)$ and $\text{max-exp}(I_2)$. The induction hypothesis gives $\text{IsMaxPos}(I_1)$ and $\text{IsMaxPos}(I_2)$, which, by definition of maximal positive BIEs, results in $\text{IsMaxPos}(\text{add}(I_1, c, I_2))$.

Dis Suppose $\text{max-exp}(I_1 \lor I_2)$. By lemma 5.23 and the induction hypothesis, we obtain $\text{max-exp}(I_1)$ and $\text{max-exp}(I_2)$.

Remains to show $\text{IsLE}(I_1)$ and $\text{IsLE}(I_2)$. We sketch the proof of $\text{IsLE}(I_1)$. Since $I_1 \lor I_2$ is representable as maximal positive, there is some BIE $J \equiv I_1 \lor I_2$ such that $\text{IsMaxPos}(J)$. From $J \equiv I_1 \lor I_2$ and lemma 5.21, we have $[\text{Zip}(J)] = [\text{Zip}(I_1 \lor I_2)]$.

For the sake of argument, assume that $I_1$ is not a classical index expression. By absence of negations, this means that it contains additional dyadic operators. Note that $\text{IsMaxPos}(J)$ implies that $J \equiv I_1 \lor I_2$ only holds if $I_1 \lor I_2$ is congruent with a BIE obtained by (partially) zipping $J$. However, zipping a maximal positive BIE to a disjunctive BIE that contains additional dyadic operators can only be done at the price of an increase of dyadic count, which would mean $[I_1 \lor I_2] > [J]$. This introduces the required contradiction since it implies $\text{Exp}(I_1 \lor I_2) < \text{Exp}(J)$ meaning that $I_1 \lor I_2$ is not maximally expanding.

Con Assume $\text{max-exp}(I_1 \land I_2)$. By lemma 5.23 and the induction hypothesis, we obtain $\text{IsMaxPos}(I_1)$ and $\text{IsMaxPos}(I_2)$. The additional properties of the subexpressions follow similarly to the previous case.

The explicit characterisation of maximal positive BIEs enforces a good spreading of the dyadic operators in the BIE. As an example, consider the constraint stating that in the arguments of disjunctions no other operators are allowed to occur. This means that all other operators must reside in other parts of the BIE. On the contrary, the dyadic operators in minimal BIEs are all placed relatively close together. In composed minimal BIEs, for example, all dyadic operators must reside in one of the subexpressions. In the next section, it is shown that the spreading of dyadic operators in maximal positive BIEs enforces exponential expansion.

Upper Bound

When zipped, every dyadic operator can cause a doubling of the number of atomic BIEs. This means that the maximal growth of the dyadic count is exponential. This upper bound is stated in the following lemma. Note that the lemma only holds for BIEs without negations.

 Lemma 5.30 If $I$ is a BIE containing $k$ disjunctions, $l$ conjunctions, but no negations, then

$$[\text{Zip}(I)] \leq 2^{k+l} - 1$$
The following theorem states that exponential growth of the dyadic count is reached for
maximal positive BIEs. This means that the upper bound of lemma 5.30 is tight. In
maximal positive BIEs, all dyadic operators are placed in such a way that ZipOr and
ZipAnd both cause exponential expansion. For a maximal positive BIE with \( k \) disjunctions
and \( l \) conjunctions, ZipOr results in a disjunction with \( 2^k \) elements. Each element contains
the \( l \) conjunctions of the original BIE. When the conjunctions in those elements are zipped
locally by ZipAnd, this results in \( 2^l \) atomic BIEs per element.

**Theorem 5.4** Let \( I \) be a maximal positive BIE containing \( k \) disjunctions and \( l \) conjunctions. Then,

\[(\text{ZipOr}) \quad \text{ZipOr}(I) = \bigvee_{1 \leq i \leq 2^k} I_i \text{ where all } I_i \text{ are free of disjunctions and contain } l \text{ conjunctions} \]

\[(\text{ZipAnd}) \quad \text{ZipAnd}(\text{ZipOr}(I)) = \bigvee_{1 \leq i \leq 2^k} \bigwedge_{1 \leq j \leq 2^l} I_{i,j} \]

**Proof:**
The proof is done by structural induction on BIEs.

**Term** For terms, the theorem is obvious.

**Add** Left to the reader.

**Dis** Suppose the BIE under consideration is of the form \( I \vee J \). By definition of maximal
BIEs, this means that \( I \) and \( J \) are classical index expressions.

**ZipOr** Since \( I \) and \( J \) are classical index expressions, we have Zip(I) = I and Zip(J) = J (by lemma 5.5). In addition, for \( I \vee J \) we have \( k = 1 \) and \( l = 0 \). Therefore,
ZipOr(\( I \vee J \)) = ZipOr(I) \( \vee \) ZipOr(J) = \( I \vee J \) is of the desired form.

**ZipAnd** By IH, ZipOr has resulted in \( \bigvee_{1 \leq i \leq 2^k} I_i \) such that all \( I_i \) contain \( l \)
conjunctions. Without loss of generality, assume that, for some integer \( x \geq 1 \), the first \( x \)
disjunctive elements constitute \( I \) and the remaining ones constitute \( J \). That is,
assume \( I \vee J \) equals \( \bigvee_{1 \leq i \leq x} I_i \vee \bigvee_{x+1 \leq i \leq 2^k} I_i \); for some \( x \geq 1 \). By IH, applying
ZipAnd to both arguments \( I \) and \( J \) results in

\[\text{ZipAnd}(I) = \bigvee_{1 \leq i \leq x} \bigwedge_{1 \leq j \leq 2^l} I_{i,j} \text{ and ZipAnd}(J) = \bigvee_{x+1 \leq i \leq 2^k} \bigwedge_{1 \leq j \leq 2^l} I_{i,j} \]

Combining these results, we find that ZipAnd(I) \( \vee \) ZipAnd(J) equals

\[\bigvee_{1 \leq i \leq 2^k} \bigwedge_{1 \leq j \leq 2^l} I_{i,j}\]

which is of the desired form.
5.7. MAXIMAL EXPANSION

Con Suppose the BIE under consideration is of the form $I \wedge J$.

ZipOr Suppose that of the $k$ disjunctions in $I \wedge J$, $k_1$ occur in $I$ and the remaining $k - k_1$ in $J$. Note that $I$ and $J$, by definition, are free of conjunctions. Then, by IH,

\[
\text{ZipOr}(I) = \bigvee_{1 \leq i < 2^{k_1}} I_i \text{ and } \text{ZipOr}(J) = \bigvee_{1 \leq i < 2^{k-k_1}} I_i
\]

Thus, the final result $\text{OrProd}(\text{ZipOr}(I), \text{ZipOr}(J))$ is a disjunction of $2^{k_1} \times 2^{k-k_1} = 2^k$ atomic BIEs.

ZipAnd Since $I$ and $J$ contain no other operators, this case is trivial.

As a direct consequence of theorem 5.4, the following corollary explicitly states the exponential number of atomic subexpressions in zipped maximal positive BIEs.

Corollary 5.5 Let $I$ be a maximal positive BIE containing $k$ nested disjunctions and $l$ nested conjunctions. Then,

\[
\text{Zip}(I) = \bigvee_{1 \leq i < 2^k} \bigwedge_{1 \leq j < 2^l} I_{i,j}
\]

where all $I_{i,j}$ are atomic.

Zipped BIEs contain one atomic subexpression more than their dyadic count. Combining this observation with the previous theorem directly leads to the following corollary.

Corollary 5.6 If $I$ is a maximal positive BIE containing $k$ nested disjunctions and $l$ nested conjunctions, then $[\text{Zip}(I)] = 2^{k+l} - 1$.

This means that maximal positive BIEs conform to the upper bound of expansion. In other words, that the upper bound is tight.

Lemma 5.31

\[
\text{IsMaxPos}(I) \Rightarrow \text{max-exp}(I)
\]

Proof: Suppose $\text{IsMaxPos}(I)$. Then, by corollary 5.6, $[\text{Zip}(I)] = 2^{k+l} - 1$. Maximal expansion of $I$ then follows by lemma 5.30.

Maximal positive BIEs thus exactly coincide with maximally expanding BIEs without negations.

Corollary 5.7 Let $I$ be a BIE representable as maximal positive and not containing negations.

\[
\text{max-exp}(I) \iff \text{IsMaxPos}(I)
\]

An example class of maximal positive BIEs is provided next, illustrating their exponential expansion.
Example: Umbrella BIEs

The following example defines so called umbrella BIEs. The name is derived from umbrella expressions which are index expressions whose form resembles an umbrella ([Bru93]). Umbrella expressions consists of a head and a number of subexpressions that consist of a single term. In umbrella BIEs, each subexpression is either a disjunction or a conjunction of two single terms. Umbrella BIEs are maximal positive BIEs in which the spreading of dyadic operators is well-illustrated. In umbrella BIEs, every dyadic operator resides in a different subexpression of the head. The example provides insight in the exponential growth of the operator count during zipping.

Example 5.9 Umbrella BIEs

A class of maximal positive BIEs are so called umbrella BIEs. The most basic of these consist of a single term head, \( k \) disjunctions and \( l \) conjunctions as subexpressions such that all subexpressions are of the form \( t \lor t' \) (\( t \land t' \)) where \( t \) and \( t' \) are single terms.

Consider the umbrella BIE (see Figure 5.13) consisting of head \( h \), \( l \) conjunctions with single-term arguments as the first \( l \) subexpressions \( I_i \), and \( k \) disjunctions with single-term arguments \( J_i \) as the last \( k \) subexpressions. The terms that constitute the arguments of subexpression \( i \) are denoted by \( t_{i_1} \) and \( t_{i_2} \). This BIE has the following structure

\[
\text{add}(\ldots \text{add}(\ldots \text{add}(h, c_1, I_1) \ldots c_l, I_l), d_{i+1}, J_{i+1}) \ldots d_{i+k}, J_{i+k})
\]

After ZipOr, we have obtained a BIE that consists of \( 2^k \) elements:

\[
\bigvee_{1 \leq i \leq k; t_i \in \{t_{i_1}, t_{i_2}\}} \text{add}(\ldots \text{add}(\ldots \text{add}(h, c_1, I_1) \ldots c_l, I_l), d_{i+1}, t_{i+1}) \ldots d_{i+k}, t_{i+k})
\]

The elements look like the one depicted in Figure 5.14. In these elements, the conjunctions reside in the leftmost \( l \) subexpressions and the other subexpressions are single terms.

By ZipAnd, each such element results in a conjunction of \( 2^l \) atomic BIEs. In total, we thus obtain a large BIE in disjunctive normal form containing \( 2^k2^l = 2^{k+l} \) atomic BIEs. This means that the dyadic count of the zipped version is \( 2^{k+l} - 1 \).
5.7.2 Maximal BIEs

The previous section described and illustrated exponential expansion caused by dyadic operators. Based on these results, this section investigates maximal expansion for BIEs that also include negations. This section therefore focuses on the expanding effect of negations.

**Explicit Characterisation**

Consider the case when ZipOr encounters a negation:

\[
\text{ZipOr}(\neg I) = \text{NotProd}(\text{Zip}(I))
\]

After zipping the subexpression, NotProd distributes the negation over its propositional structure and restores the DNF. Restoring the DNF increases the dyadic count if the subexpression contains both disjunctions and conjunctions. For BIEs of special form, this is formally described in the following lemma.

**Lemma 5.32** Consider a BIE \( I \) in DNF with the same number of conjunctions in each element of the disjunction:

\[
I = \bigvee_{1 \leq i \leq x} \bigwedge_{1 \leq j \leq y} I_{i,j}
\]

for some \( x, y \geq 1 \). Then

\[
\text{NotProd}(I) = \bigvee_{1 \leq k \leq y'} \bigwedge_{1 \leq i \leq x} I_{i,k,j_i}
\]

**Proof:**

\[
\text{NotProd}(\bigvee_{1 \leq i \leq x} \bigwedge_{1 \leq j \leq y} I_{i,j}) = \text{OrProd}'(\bigvee_{1 \leq j \leq y} \neg I_{1,j}, \ldots, \bigvee_{1 \leq j \leq y} \neg I_{x,j})
\]
This results in \( x - 1 \) calls to the original \text{OrProd}:

\[
\text{OrProd}( \bigvee_{1 \leq j \leq y} I_{1,j}, \ldots, \text{OrProd}( \bigvee_{1 \leq j \leq y} I_{x-1,j}, \bigvee_{1 \leq j \leq y} I_{x,j}) )
\]

Each call to \text{OrProd} multiplies the number of arguments in the result. Finally, this results in a BIE in DNF with \( y^x \) elements in the disjunction that each contain \( x \) atomic elements in conjunction. Namely, all possible \( y^x \) conjunctions are formed that contain exactly one atomic BIE out of each of the \( x \) original disjunctions.

The expanding effect of negations is the number of dyadic operators their subexpression contains. This means that maximally expanding BIEs are such that all negations have all disjunctions and conjunctions in their subexpression. Figure 5.15 gives a schematic overview of this situation. All disjunctions and conjunctions reside in the so called dyadic subexpression. All negations lie on the path, called the negation path, connecting the head with the dyadic subexpression. If the dyadic subexpression is maximal positive, i.e. is maximally expanding by absence of negations, then maximal expansion is reached for the complete BIE.

![Figure 5.15: Schematically Denoted Maximal BIE.](image)

The definition below describes the part of Figure 5.15 that is complementary to the dyadic subexpression. It makes sure that the negations form a path towards a subexpression that contains all dyadic operators. This subexpression is to be maximal positive as defined in definition 5.14.

**Definition 5.16 Maximal BIEs**

**Term** All terms are maximal BIEs.

**Neg** \( \neg I \) is maximal if \( I \) is maximal

**Add** \( \text{add}(I, c, J) \) is maximal if...
5.7. MAXIMAL EXPANSION

left simple $I$ is a classical IE and $J$ is maximal, or

right simple $J$ is a classical IE and $I$ is maximal, or

both maximal $I$ and $J$ are maximal positive (as in definition 5.14).

A maximal BIE can be of three forms. First, terms are maximal. Second, a BIE of the form $\neg I$ is maximal if $I$ is maximal. This describes the negation path. Composed BIEs may, in turn, be of three forms. In order to form a correct negation path leading to a single subexpression with all dyadic operators, composed maximal BIEs have one subexpression that is free of logical operators while the other recursively is maximal. This covers cases left simple and right simple. The final case of composed BIEs states that both subexpressions of a composed maximal BIE may also be maximal positive BIEs.

Upper Bound

The expansion of maximal BIEs defines an upper bound on the expansion of BIEs in general. We now focus on expansion that results from evaluating negations. The expanding effect of negations is caused by several calls to OrProd, as examined in lemma 5.32. Series of negations, such as those on a negation path, cause a repetition of this effect.

This repeated effect of series of negations is captured in the function NegMax, which is provided below. This function describes the number of atomic BIEs in the zipped result of maximal BIEs. As defined in the previous section, the parts of maximal BIEs that cause expansion are the negation path and the dyadic subexpression. The parameters of NegMax describe this information. The first two parameters $x$ and $y$ describe the form of the zipped dyadic subexpression. The first parameter $x$ denotes the number of elements in the disjunction. The second parameter $y$ denotes the number of atomic BIEs within these elements. In other words, the zipped maximal positive subexpression is of the form

$$\bigvee_{1 \leq i \leq x} \bigwedge_{1 \leq j \leq y} I_{i,j}$$

The function only works for dyadic subexpressions that contain the same number of conjunctions in each element of the disjunction. Since the dyadic subexpression is maximal positive, the numbers $x$ and $y$ may be obtained by theorem 5.4. The remaining parameter $m$ describes the negation path by the number of negations on it.

The result of a call $\text{NegMax}(x, y, m)$ represents the number of atomic BIEs after zipping the maximal BIE. The result is a pair $(a, b)$, of which $a$ denotes the number of elements in the disjunction of the zipped BIE and $b$ denotes the number of atomic BIEs within each such element. The product $a \times b$ thus denotes the number of atomic BIEs in the zipped maximal BIE.

Definition 5.17 Consider a maximal BIE with $m$ negations and a dyadic subexpression that, after zipping, consists of a disjunction of $x$ elements that all contain $y$ atomic BIEs.
Then, $\text{NegMax}(x, y, m)$, as defined below, yields a pair $(a, b)$ where $a$ equals the number of disjunctive elements in the zipped result and $b$ denotes the number of atomic BIEs in such elements.

$$\text{NegMax}(x, y, m) = \begin{cases} (x, y) & \text{if } m = 0, \\ \text{NegMax}(y^x, x, m - 1) & \text{otherwise}. \end{cases}$$

Function $\text{NegMax}$ iteratively computes the expansion of an $m$-times negated maximal positive BIE. Starting with $x$ elements in the disjunction each containing $y$ atomic BIEs, the first negation results in a disjunction of $y^x$ elements that each contain $x$ atomic BIEs (see lemma 5.32). These resulting values are then input for the remaining $m - 1$ negations. Function $\text{NegMax}$ is illustrated by the following example.

**Example 5.10** Consider the BIE $I = \text{add}(t \land c, t \lor t)$. It contains a single disjunction and a single conjunction. After zipping, a BIE is obtained with elements in disjunction $(x = 2)$ that both consist of two atomic BIEs in conjunction $(y = 2)$. The dyadic count after zipping is three: $\text{NegMax}(2, 2, 0) = (2, 2)$ thus the dyadic count is $2 \times 2 - 1$.

Now consider the negated form of $I$, i.e. $\neg I$. Zipping $\neg I$ involves applying $\text{NotProd}$ to $\text{Zip}(I)$. This is modeled by $\text{NegMax}(2, 2, 1) = \text{NegMax}(2^2, 2, 0) = (4, 2)$. Having zipped $\neg I$, we have thus obtained a disjunction of $4$ elements each consisting of $2$ atomic BIEs. In total, $\text{Zip}(\neg I)$ contains $4 \times 2 = 8$ atomic BIEs and has an dyadic count of $7$.

Negating $I$ twice leads to $\text{NegMax}(2, 2, 2)$. By repeating the computation, this results in $\text{NegMax}(4, 2, 1) = \text{NegMax}(2^4, 4, 0) = (16, 4)$. $\text{Zip}(\neg \neg I)$ thus contains $16 \times 4 = 64$ atomic BIEs and its dyadic count is 63.

Negating $I$ three times results in $\text{NegMax}(2, 2, 3) = \text{NegMax}(16, 4, 1) = (4^{16}, 16)$. This means that $\text{Zip}(\neg \neg \neg I)$ contains $16 \times 4^{16} \approx 6.87 \times 10^{10}$ atomic BIEs.

The dyadic count thus grows extremely fast if negations are expanded. The practical consequences, however, might not be very severe. First of all, we do not expect users to formulate queries of the form $\neg \ldots \neg I$. In addition, the query formulation tools described in section 5.2.3 can be exploited to form non-problematic BIEs. Furthermore, the extreme expansion of negations can be prevented by using other forms of zipping. This means that the current expanding effect of negations, due to the elementary definition of the zipping functions, can be overcome. This is further elaborated on in section 5.7.4.

The following lemma exploits function $\text{NegMax}$ in proving the expansion of maximal BIEs. A maximal BIE with $k$ disjunctions, $l$ conjunctions, and $m$ negations contains a maximal positive BIE with all the dyadic operators. When zipped, the maximal positive subexpression results in a disjunction of $2^k$ elements each containing $2^l$ atomic BIEs. To describe the expansion of the complete maximal BIE, function $\text{NegMax}$ is called with these parameters.
Lemma 5.33 Let $I$ be a maximal BIE containing $k$ disjunctions, $l$ conjunctions, and $m$ negations. Then

$$[\text{Zip}(I)] = x \times y - 1$$

where $(x, y) = \text{NegMax}(2^k, 2^l, m)$.

Proof:
The proof is done by structural induction on BIEs. The basis, i.e. the case for terms, is trivially omitted.

Neg Suppose the BIE under consideration is of the form $\neg I$. By IH, we have $[\text{Zip}(I)] = x' \times y' - 1$ where $(x', y') = \text{NegMax}(2^k, 2^l, m - 1)$. Thus, Zip($I$) is in DNF and consists of a disjunction of $x'$ elements that are conjunctions of $y'$ atomic BIEs. By lemma 5.32, we then obtain that Zip($\neg I$) has $y'y'$ disjunctions that all consist of $x'$ conjunctions. Thus, $[\text{Zip}(\neg I)] = (x, y) - 1$ where $(x, y) = \text{NegMax}(2^k, 2^l, m)$.

Composed Suppose the BIE under consideration is of the form $\text{add}(I, c, J)$. There are three possibilities:

left simple Suppose that $I$ is a classical index expression, thus $[I] = 0$. By IH, we have that $[\text{Zip}(J)] = x \times y - 1$, where $(x, y) = \text{NegMax}(2^k, 2^l, m)$. Since $I$ contains no operators, the number of operators of the result of OrCons is equal to the number of operators in $J$. Thus, $[\text{Zip}($add$\langle I, c, J \rangle)] = x \times y - 1$.

right simple Similar to left simple.

both maximal If $I$ and $J$ are maximal positive BIEs and together contain $k$ disjunctions and $l$ conjunctions, we have, by theorem 5.4, that $[\text{Zip}($add$\langle I, c, J \rangle)] = 2^{k+l} - 1$. This is $x \times y - 1$ for $(x, y) = \text{NegMax}(2^k, 2^l, 0)$.

Lemma 5.33 provides a tight upper bound on the expansion of BIEs. As a corollary, we state the boundary cases of this lemma. That is, we examine maximal BIEs that do not contain disjunctions, conjunctions, or negations, respectively.

Corollary 5.8 no disjunctions. Consider a BIE $I$ that is maximal without disjunctions, i.e. $k = 0$. Negating $I$ does not affect its atomic size: $[\text{Zip}(\neg I)] = x \times y - 1$, where $(x, y) = \text{NegMax}(1, 2^l, m)$. The reason for this is that for any $m$ we have that $\text{NegMax}(1, 2^l, m)$ is a constant.

no conjunctions. The examination of maximal BIEs without conjunctions is done similarly to the case no disjunctions.

no negations. Consider a BIE $I$ that is maximal without negations, i.e. $m = 0$. This means the BIE under consideration is a maximal positive BIE. Then, $\text{NegMax}(2^k, 2^l, 0) = (2^k, 2^l)$. This leads to an dyadic count of $2^k2^l - 1 = 2^{k+l} - 1$ which corresponds to the upper bound for maximal positive BIEs (corollary 5.6).

The example in the next subsection illustrates that the expansion of maximal BIEs.
Example: Negated Umbrella BIEs

Simple maximal BIEs are formed by negating an umbrella BIE a number of times. Such BIEs are called negated umbrella BIEs.

**Example 5.11 Negated Umbrella BIEs** Let \( I \) be an umbrella BIE as defined in example 5.9. Consider BIEs of the form \( \neg \ast I \), where \( \ast \) denotes zero or more occurrences (Kleene star). Such BIEs are called negated umbrella BIEs. Negated umbrellas are maximal since their dyadic subexpression, the umbrella BIE, is maximally expanding and the negations form a negation path.

Consider a negated umbrella BIE with \( m \) negations, \( k \) disjunctions, and \( l \) conjunctions. Zipping this negated umbrella BIE leads to

\[
\text{Zip}(\neg \ldots \neg I) = \text{NotProd}(\ldots \text{NotProd}(\text{Zip}(I)))
\]

where, since \( I \) is an umbrella BIE, \( \text{Zip}(I) = \bigvee_{1 \leq i \leq 2^k} \bigwedge_{1 \leq j \leq 2^l} I_{i,j} \). That is, the zipped umbrella BIE consists of a conjunction of \( 2^k \) elements that each contain \( 2^l \) atomic BIEs in the conjunction. In order to compute the dyadic count of the resulting BIE, the \( m \) negations must now be taken into account. This means that the dyadic count of the resulting BIE is \( x \times y - 1 \), where \( (x, y) = \text{NegMax}(2^k, 2^l, m) \).

### 5.7.3 Quantitative Analysis of Maximal BIEs

As for minimal BIEs, we performed experiments for computing the fraction of maximal BIEs

\[
\text{MaxFrac}(t, d, c, n) = \frac{n\text{Max}(t, d, c, n)}{n\text{BIE}(t, d, c, n)}
\]

In section 5.7.3, we investigate the influence of negations on the fraction of maximal BIEs. In section 5.7.3, the influence of dyadic operators on this fraction is studied.

**Negations**

As in the experiments for minimal BIEs, we generated all possible BIEs with seven terms, one disjunction, two conjunctions, and a varying number of Again, we considered zero to six negations. That is, we investigated \( n\text{Max}(7, 2, 2, n) \), for \( 1 \leq n \leq 6 \). The results are provided in Figure 5.16.

On the x-axis of Figure 5.16, the number of negations \( n \) is plotted. As before, the line through the full circles denotes the growth of the total number of BIEs \( n\text{BIE}(7, 2, 2, n) \). This line is to be seen in the light of the right y-axis. The left y-axis provides percentages and, as the right one, has a logarithmic scale. The line through the crosses is to be seen in the light of this y-axis. The crosses denote percentages of maximal BIEs, i.e. \( 100 \times \text{MaxFrac}(7, 2, 2, n) \).
5.7. MAXIMAL EXPANSION

The fraction of maximal BIEs is small and rapidly decreases with an increase of the number of negations. It ranges from 6.67 percent for BIEs without negations, via 0.65 percent for a single negation and 0.03 percent for three negations, to 0.001 percent for six negations.

Dyadic Operators

In order to investigate the influence of dyadic operators on the fraction of maximal BIEs, all BIEs with a given number of terms and varying numbers of dyadic operators were generated. Since for BIEs with seven terms the fraction of maximal BIEs is very low, we generated all maximal BIEs with 11 terms. Still the fraction of maximal BIEs is not high, especially for more than five dyadic operators. Therefore, the results, shown in Figure 5.17, only include up to four dyadic operators.

In Figure 5.17, the y-axis denotes percentages. The y-value of a dot labeled by a pair \((d, c)\) represents \(\text{MaxFrac}(11, d, c, 0)\). Lines are drawn through dots that represent the same dyadic count. The x-axis denotes the difference in numbers of disjunctions and conjunctions. A dot \((d, c)\) has x-value \(d - c\).

As stated before, only a small percentage of BIEs is maximal. This is illustrated by the low y-values of most of the dots in the figure. The small fraction of maximal BIEs is due to the strictness of the constraints in their definition.

In absence of one type of dyadic operators, the constraints they impose are equally restrictive. This is illustrated by equal y-values of dots \((0, i)\) and \((i, 0)\), meaning \(\text{MaxFrac}(11, 0, i, 0) = \text{MaxFrac}(11, i, 0, 0)\). By definition, the arguments of disjunctions in maximal positive BIEs
are not allowed to contain additional operators. Also by definition, the arguments of conjunctions do not contain additional conjunctions. In absence of disjunctions, this means that the arguments do not contain additional operators.

The lines in Figure 5.17 first move upward and then downward again. This means that co-occurrence of both types of dyadic operators allows more maximal BIEs than when one type of dyadic operators prevails.

### 5.7.4 Dealing with Impractical Expansion

Zipping, as we defined it, can expand BIEs to impractical sizes. An illustration of this, clearly showing the expanding effect of negation paths, was given in example 5.10. This section addresses three issues concerning impractical expansion. First, an example illustrates that many complex information needs can still be compactly represented by BIEs that do not zip to impractical sizes. Second, we describe how the formulation tools of section 5.2.3 can be exploited for formulation non-problematic BIEs. Third, we sketch a way of handling negations in zipping that eliminates their exploding effect on expansion.

**Example 5.12** Consider a personal computer with 128MB of memory. We investigate the complexity of maximal positive BIEs that do not exceed the amount of memory when zipped. In particular, we analyse how many constituents such BIEs may have. Assume that terms and connectors are represented with 4 bytes each. The zipped result features \(2^{k+l}\) atomic BIEs of size \(16t(t-1)\). This means that maximal positive BIEs may consist of \(t\) terms, \(k\) disjunctions, and \(l\) conjunctions such that

\[
16t(t-1) \cdot 2^{k+l} \leq 2^{27}
\]

Isolating the number of terms, this becomes

\[
t(t-1) \leq 2^{27-k-l}
\]
We investigate for which numbers of terms and dyadic operators the equation holds. In Figure 5.18 the line \( t(t-1) = 2^{23-k-1} \) is plotted through the dots. All combinations of \( t \) terms and \( k+1 \) dyadic operators that lie under this line satisfy the equation. In other words, their zipped versions fit in memory. All combinations that lie above the line are not viable.

![Graph showing viable area for maximal positive BIEs](image)

Figure 5.18: Viable area for maximal positive BIEs.

For a combination of terms and dyadic operators to be viable, there is a second constraint. In valid BIEs, the number of terms is larger than the number of dyadic operators. If this is not the case, there are not enough terms to fill the arguments of the dyadic operators. The other, dashed, line in Figure 5.18 plots the line \( t = k+1 \). This means that the area left of this line is not viable by a lack of terms.

Concluding, the area right of the dashed line and under the line through the dots gives all viable combinations. For example, maximal positive BIEs with 16 terms are allowed to contain 15 dyadic operators. Maximal positive BIEs with 100 terms can contain 10 dyadic operators. For 200 terms, 8 dyadic operators may be contained. This means that many complex information needs can be represented compactly by BIEs. In other words, for practical application in IR and IF, maximal positive BIEs are tractable.

**Preventing Maximal BIEs**

The formulation tool of section 5.2.3 combines navigating in a hyperindex for index expressions with actions to construct BIEs. Problematic BIEs can be prevented by signaling the user if he is about to construct such a BIE. Another way of prevention is not to present actions or arguments that would lead to the construction of problematic BIEs. Identifying problematic BIEs can be done with a function that yields the required number of constituents after zipping. Such a function could be a generalisation of \texttt{NegMax}. Applying a suitable threshold on the expansion yields the desired effect.
Formulation information needs by relevance feedback (see section 5.2.3) allows good control over the form of the query. Relevance feedback positively and negatively identifies (parts of) documents as relevant. Characterising documents with index expressions means that the part of the representation created by relevance feedback (see Figure 5.4) does not contain nested dyadic operators. This ensures good tractability.

Navigational query formulation has proven an effective approach for index expressions. Based on the subexpressions of some initial set of index expressions, a network is created that allows structural traversal of links. If such networks were available for BIEs, similar navigational techniques could be applied. This requires a notion of subexpressions for BIEs. BIEs can be broken down in their components by relation HasComp (see def. 5.5). This may provide a starting point for a more elaborate subexpression relation. Problematic BIEs can then be excluded from the network, possibly replacing them by similar but non-problematic ones. Currently, this is an issue for further research.

**Variant of Zipping**

The essence of zipping is evaluating logical operators without altering the intended semantics. The exploding effect on expansion of negations can be eliminated by partial evaluation. In this variant of zipping, dyadic operators are fully evaluated but negations are not. The new rules concerning negations for ZipOr and ZipAnd are stated as

\[
\text{ZipOr}(\neg I) = \neg \text{ZipOr}(I) \quad \text{ZipAnd}(\neg I) = \neg \text{ZipAnd}(I)
\]

Substituting these new rules for negations in the zipping procedure, disjunctions and conjunctions are zipped up until they meet a negation. Then, the negation is not zipped downward, or, evaluated. This will cut down the size of the result drastically since NotProd is not called. As a consequence, less strict normalisation is reached.

The reason for this is that the zipped result has a uniform recursive structure. It starts with a number of negations. Then, like in DNF, a disjunction of conjunctions is found. However, the subexpressions are, unlike DNF, not atomic, but recursively of the same form. Concretely, the form of the zipped result is

\[
\neg \bigvee_{1 \leq i \leq x} \bigwedge_{1 \leq j \leq y} I_{i,j}
\]

where the priorities of the dyadic operators in the elements \( I_{i,j} \) are properly evaluated. This eases the design of matching techniques.

For this alternative zipping function, the growth in dyadic count caused by zipping will be at most the growth reached for maximal positive BIEs. This means that the viable area of Figure 5.18 also applies to maximal BIEs (containing negations) and the alternative zipping function.
5.8 Applying BIEs in ID

Formulating an information need with a BIE was described in section 5.2.3. In fact, a general setting for manipulating BIEs was provided. This section concentrates on the remaining two cardinal functions for descriptors in ID: characterising documents with BIEs and computing similarity between BIEs. It is shown that the difficulties in obtaining full-fledged BIEs from text is bypassed by including normalisation (as described in section 5.4) in matching. Concretely, this means that obtaining classical index expressions from text is sufficient to retrieve documents for queries stated as BIEs.

5.8.1 Obtaining BIEs from Text

It is a hard linguistic problem to correctly find out the intended meaning of textual constructs that involve conjunctions\(^1\). Conjunctions are common linguistic constructs ([Sme92]). However, discovering the correct logical structure and scopes of the arguments is a difficult task. The scope of an operator defines the bounds of its arguments. A related issue, hinging on the priorities assigned to logical operators, is to ascertain the intended nesting structure in the presence of multiple operators. Furthermore, the meaning of logical constructs in natural language often does not coincide with a (direct) Boolean interpretation.

In the field of natural language processing, much work is devoted to designing powerful parsers. For example, probabilistic parsing ([Bod89]) generates all possible parse trees accompanied by their “probability” of correctness. However, this potentially leads to extremely large numbers of parse trees to be considered. A two-stage conjunction reduction yielding structured concepts is described in [Olt00]. For overviews of the use of NLP techniques and resources in ID, the reader is referred to [SCM95, Str95, Sme97].

We conclude that automatically obtaining BIEs from text in its full variety of logical constructs is not supported by current NLP techniques. Despite this, the next section shows that the structure in queries formulated as BIEs does not have to be lost in matching with document contents. The reason for this is that indexing documents with IEs, as described in Section 4.7.2, is sufficient for our purposes.

5.8.2 Matching BIEs

Comparing queries or profiles with document content is essential for ID. The availability of numerical similarity measures for BIEs enables their use in many important ID tasks. For instance, ID tasks that involve similarity measures are document ranking, classification, routing, and clustering.

Assume that the information need at hand is formulated as a BIE, possibly containing all types of operators. Formulating information needs with BIEs was discussed in section

\(^{1}\)Here, the term conjunctions covers linguistic constructs containing the word *and* or *or*. 
5.2.3. The previous subsection suggested that documents are better indexed with classical IEs than with full-fledged BIEs. Indexing documents with IEs was discussed in Section 4.7.2. Therefore, assume the document characterisation is a set of IEs. Matching a BIE with a set of IEs can be done by viewing the characterisation as a conjunction of IEs. In this way, the document characterisation is (implicitly) specified as a BIE. A similar approach is taken for the Boolean keyword approach, where the set of keywords representing a document is also considered in conjunction during matching.

Comparing BIEs is eased by the normalisation function as described in section 5.4. The fact that we can bring arbitrary BIEs into normal form allows us to exploit matching functions for IEs with only minor modifications. The DNF normal form delivers a (logical) sum of products of atomic BIEs. The sum and product are directly translated to mathematical sum and product and do not involve any particular matching function. This implies that only atomic index expressions have to be directly compared, meaning that similarity measures for IEs only have to be extended for the case of negations.

Matching BIEs by exploiting normalisation is schematically depicted in Figure 5.19. The similarity measure for BIEs $\text{sim}_{\text{BIE}}$ is defined as

$$
\text{sim}_{\text{BIE}} = \text{sim}_v(Zip(I), Zip(J))
$$

We first exploit the overall disjunctive structure of DNF by dealing with the disjunctions $Zip(I) = \bigvee_{i=1}^{k} I_i$ and $Zip(J) = \bigvee_{j=1}^{l} J_j$.

$$
\text{sim}_v(I, J) = \sum_{i=1}^{k} \sum_{j=1}^{l} \text{sim}_A(I_i, J_j)
$$

Next, observe that the arguments of $\text{sim}_A(I, J)$ are conjunctions $I = \bigwedge_{i=1}^{k} I_i$ and $J = \bigwedge_{j=1}^{l} J_j$ of atomic BIEs. This implies that we can compute their similarity as a double product of elementary similarities for atomic index expressions. We denote the similarity measure for IEs by $\text{sim}$. 

![Figure 5.19: Matching BIEs with Normalisation.](image-url)
The functions \( \text{sim}_v \) and \( \text{sim}_\wedge \) cater for disjunctions and conjunctions, respectively. Therefore, the basic similarity measure for IEs \( \text{sim} \) only needs to be augmented for negations. We do this by adding two simple rules as provided in Equation 5.3.

\[
\begin{align*}
\text{sim}(\neg I, J) &= 1 - \text{sim}(I, J) \\
\text{sim}(I, \neg J) &= 1 - \text{sim}(I, J)
\end{align*}
\] (5.3)

The rules of Equation 5.3 support the following simple lemma.

**Lemma 5.34**

\[
\begin{align*}
\text{sim}(\neg\neg I, J) &= \text{sim}(I, J) \\
\text{sim}(\neg I, \neg J) &= \text{sim}(I, J)
\end{align*}
\]

Matching functions for IEs were described and analysed in section 4.6. All of these can be transformed into similarity measures for BIEs by the above process. This is due to their general set-up by pattern matching. Properties of \( \text{sim}_{\text{BIE}} \) may thus hinge on the similarity measure for IEs that is used. Several general statements about the similarity measure for BIEs are made below. The following lemma states that equivalent BIEs effectuate the same similarity scores.

**Lemma 5.35**

\[
I \equiv J \Rightarrow \forall K : \text{sim}_{\text{BIE}}(I, K) = \text{sim}_{\text{BIE}}(J, K)
\]

The next lemma shows that symmetry of the similarity measure for IEs is preserved in the transformation to a similarity function for BIEs.

**Lemma 5.36** If the similarity measure for IEs used is symmetric, then

\[
\text{sim}_{\text{BIE}}(I, J) = \text{sim}_{\text{BIE}}(J, I)
\]

The next lemma indicates that disjunctive BIEs show additive similarity.

**Lemma 5.37**

\[
\text{sim}_{\text{BIE}}(I_1 \lor I_2, J) = \text{sim}_{\text{BIE}}(I_1, J) + \text{sim}_{\text{BIE}}(I_2, J)
\]
The following example illustrates the workings of the similarity measure for BIEs.

**Example 5.13** Consider the following BIE as query

\[ q = \text{add}(\text{retrieval} \lor \text{filtering, of, images} \land \text{documents}) \]

Furthermore, assume a certain document is indexed by the following index expressions (which are denoted in conjunction for convenience)

\[ d = \text{add}(\text{filtering, of, documents}) \land \text{add}(\text{retrieval, of, images}) \]

The estimated relevance of the document to the query, as computed by their similarity \( \text{sim}_{\text{BIE}}(q, d) \), is

\[
( \text{sim}(\text{add}(\text{retrieval, of, images}), \text{add}(\text{filtering, of, documents})) \\
\times \text{sim}(\text{add}(\text{retrieval, of, images}), \text{add}(\text{retrieval, of, images})) \\
\times \text{sim}(\text{add}(\text{retrieval, of, documents}), \text{add}(\text{filtering, of, documents})) \\
\times \text{sim}(\text{add}(\text{retrieval, of, documents}), \text{add}(\text{retrieval, of, images})) ) \\
+ \\
( \text{sim}(\text{add}(\text{filtering, of, images}), \text{add}(\text{filtering, of, documents})) \\
\times \text{sim}(\text{add}(\text{filtering, of, images}), \text{add}(\text{retrieval, of, images})) \\
\times \text{sim}(\text{add}(\text{filtering, of, documents}), \text{add}(\text{filtering, of, documents})) \\
\times \text{sim}(\text{add}(\text{filtering, of, documents}), \text{add}(\text{retrieval, of, images})) )
\]

The actual similarity value depends on the used similarity measure for index expressions.

The next section illustrates that many functions on BIEs are readily implemented by pattern matching in a functional language.

### 5.9 Implementation

We have implemented most of the functionality of BIEs that was presented in this chapter. For example, the matching component of the PROFILE project (see Chapter 3) is capable of computing similarity values for BIEs based on the procedure given in the previous subsection. In addition, the examination of the fractions of minimal and maximal BIEs, as provided in sections 5.6.4 and 5.7.3, exploited generator algorithms and functions to check the type of BIE at hand.

Our implementation of BIEs is written in a functional language. Functional languages are suitable for our purposes since they allow for pattern matching and are based on rewriting rules. This enabled us to readily implement the functionality of BIEs on a suitable level of abstraction. The selected functional language is CLEAN ([PE98]), developed at the University of Nijmegen. However, other functional languages may be equally suitable.
BIEs are defined in CLEAN by an algebraic type definition BIE. Constructors are used to identify the five possible forms of BIEs. The constructor Term denotes single terms, Add is used for composed BIEs, Dis for disjunctions, Con for conjunctions, and Neg for negated BIEs. The predefined type String is used to represent terms and connectors.

\[
\texttt{BIE} = \texttt{Term} \ 	exttt{String} \\
\text{Add} (\texttt{BIE}) \ 	exttt{String} (\texttt{BIE}) \\
\text{Dis} (\texttt{BIE}) \ (\texttt{BIE}) \\
\text{Con} (\texttt{BIE}) \ (\texttt{BIE}) \\
\text{Neg} (\texttt{BIE})
\]

Our implementation consists of five modules. Figure 5.20 sketches the interdependencies between these modules. The most basic module is BIE, providing elementary functionality of BIEs. Other modules import this basic module, which is reflected in the figure by solid arrows. Dashed arrows show practical interdependencies. For example, the Generator module is used for experimenting with the Normal module. The modules are discussed in more detail below.

**Figure 5.20: Modules in Implementation.**

**BIE** This module provides the elementary functionality for BIEs. Next to the type definition of BIEs, it contains routines to compute the number of terms and connectors and the head of a BIE (generalised versions of the functions provided in Figure 4.10). In addition, it provides algorithms to count the number of operators in BIEs, i.e. algorithms to compute DisCount and ConCount (definition 5.4) and a similar algorithm for the number of negations. Furthermore, it supports check functions for equivalence (def. 5.7), minimality, as expressed by IsMin (def. 5.13), maximality, denoted by IsMax (def. 5.16), and functions to identify if a BIE is maximal positive, specified by IsMaxPos (def. 5.14), or atomic (according to def. 5.2). Finally, it contains an algorithm to compute the twigs of an IE and a defoliation procedure (Section 4.3).
Normal This module covers the syntactical normalisation of BIEs. It contains the zip functions ZipAnd and ZipOr, including implementations of all the auxiliary routines used in zipping such as OrProd and AndCons (Section 5.4.3).

Sim This module offers the similarity functions for classical index expressions that were described in chapter 4.6 and the similarity scheme for BIEs, as described in section 5.8.2. For the latter, it requires normalisation functions, as supported by the module Normal.

Generator This module supports generator functions for classical index expressions (Section 4.7.3) and BIEs. The functions, provided with the number of required constituents, deliver all (B)IEs that contain exactly these constituents. In addition, it contains a subexpression generator for index expressions (Section 4.5). The functionality offered by the Generator module is exploited to experiment with other modules. For instance, the evaluation of similarity functions described in Section 4.6.5 used the subexpression generator.

Count This module offers functions to count several types of (B)IEs with given numbers of constituents. This includes, for example, functions to compute the number of atomic BIEs with given numbers of terms and negations. In addition, it supports functions nMin, nMax, and nBIE that were used in the quantitative analyses of sections 5.6.4 and 5.7.3. These functions do not generate all possible (B)IEs but, more efficiently, compute the totals directly.

To illustrate the direct translation between the functionality as defined in this chapter and the CLEAN syntax, we provide a function to check if a BIE is minimal.

Example 5.14 This example provides the implementation of the predicate IsMin (Def. 5.13). Function IsMin has a BIE as argument and produces Boolean output. It uses auxiliary functions to check if a BIE is free of dyadic operators (DyadicFree), free of disjunctions (DisFree), and free of conjunctions (ConFree). In CLEAN, && and || denote dis- and conjunctions, respectively.

\[
\begin{align*}
\text{IsMin} &: \text{BIE} \rightarrow \text{Bool} \\
\text{IsMin} (~\text{Term}\ x~) &= \text{True} \\
\text{IsMin} (~\text{Add}\ i\ c\ j~) &= \text{IsMin} i && \text{IsMin} j && (~\text{DyadicFree}\ i\ ||\ \text{DyadicFree}\ j~) \\
\text{IsMin} (~\text{Dis}\ i\ j~) &= \text{IsMin} i && \text{IsMin} j \\
\text{IsMin} (~\text{Con}\ i\ j~) &= \text{IsMin} i && \text{IsMin} j && \text{DisFree} i && \text{DisFree} j \\
\text{IsMin} (~\text{Neg}\ i~) &= \text{IsMin} i && (~\text{DisFree}\ i\ ||\ \text{ConFree}\ i~)
\end{align*}
\]

The implementation gives the core functionality of BIEs. For applications in ID, it should be augmented with a tool to formulate information needs (see section 5.2.3) and procedures to index documents with index expressions (see section 4.7.2).
5.10 Outlook

The theoretical foundation developed in this chapter is more than sufficient to justify a prototype information system based on BIEs. The usefulness of BIEs in the formulation of (complex) information needs can then be validated. Important issues in the design of the prototype are well-supported interaction and an iterative formulation process.

Zipping dyadic operators upward, as examined in this chapter, results in a logical combination of atomic BIEs. The dual approach is to zip the dyadic operators downward. This results in a compact representation since subexpression sharing is effectuated by nesting the dyadic operators. This dual form of zipping, aiming at nesting as many operators as possible, is an NP-complete problem. The reason for this is that minimizing Boolean functions is NP-complete ([Weg87]) and BIEs form a superset of Boolean propositions.

Additional research must show if navigational formulation mechanisms known for classical index expressions can be exploited for BIEs. For navigational query formulation, networks of BIEs should be constructed. Concerning this, three major challenges can be identified: automatically obtaining an initial set of BIEs, the definition of a suitable navigation relation for BIEs, and redundancy due to syntactical variation.

Interesting research may be directed towards the design of BIEs with soft operators. The p-norm model, as described in section 2.2, may serve as a basis for this. It will be interesting to see if the promising results for the p-norm keyword model, as reported in [Sav94], can also be obtained for BIEs.
Chapter 6

INdex Navigator

This chapter describes a dynamic hypertext system for the WWW based on index expressions, the INdex Navigator (INN). The INN implements a dynamic form of the stratified architecture for index expressions, allowing syntactically enabled navigation for dynamic information environments. The INN is available on the WWW. The work reported on in this chapter resulted from cooperation with Mark van Uden ([Ude99]) and Pim van Mun ([Mun99]).

This chapter has the following structure. Section 6.1 explains the advantages of navigational query formulation. Section 6.2 compares the INN with other systems. Section 6.3 elaborates on the stratified architecture for index expressions. Section 6.4 illustrates Query by Navigation. Section 6.5 describes the workings of the INN system. Section 6.6 illustrates some experiences with the INN. Section 6.7 shows how the INN can be augmented with persistent user information. Finally, Section 6.8 provides directions for further research.

6.1 Navigational Query Formulation

Searching information from a large and dynamic information space, the ultimate example of which is the WWW, causes several serious difficulties. A number of these concern query formulation. According to [BOB82, Swa88], a major problem is (caused by) the inherent vagueness of information needs. Therefore, formulating the information need concisely, without an explicit description of the expected result, is very difficult. In order to increase the user’s knowledge about the field of interest, ID systems should enable users to explore topics of interest. Related to an increase in knowledge are shifts in interests ([BOB82]). Retrieval systems should thus support interactive reformulation. A third major problem concerns constructing (syntactically) correct and (semantically) sensible complex descriptors ([OP98, Rag96]). Letting the user select descriptors from a set of (correct) options can bypass this problem. Fourth, broad queries often result in low precision. IR systems should therefore aim at preventing imprecise queries.
Formulating queries by navigation in an abstraction of the information space eases the problems mentioned ([LZ93, Bru93, WF96, WC91]). To this end, stratified architectures have been developed, containing an ancillary layer that forms an abstract description of the contents of the information space ([AGM92, Bru93]). This meta layer can provide an overview of the concepts present. This helps users in exploring their field of interest. Searchers can then formulate their need by recognizing rather than formulating relevant concepts. The vagueness in the information need can be further decreased by concept exploration: by inspecting actual documents that correspond to a concept. In this way, the user can learn what the concept means. This is the second way in which navigation assists exploration. Since the IR system generates the overview, it can guarantee correctness and sensibility of the descriptors offered. To this end, for instance, the descriptors can be taken from available documents. Shifts in interest are naturally supported by navigation through selection of a different direction in the overview. Finally, navigational formulation techniques enable users to iteratively select more specific descriptors. In general, this eases the way to descriptors of proper specificity which is very important in rich domains such as the WWW. Concluding, we advocate the combination of searching and exploration based on navigation in an ancillary structure.

However, the size and dynamics of the WWW imply that a complete and up to date abstraction cannot be constructed. Consequently, navigational query formulation in ancillary layers is not directly applicable to these information spaces.

In this chapter, we show how the WWW can instead be abstracted and navigated by using structured descriptors. We developed the INdex Navigator (INN), a dynamic information system for query formulation on and exploration of the WWW. The INN is based on Query by Navigation (QBN). QBN ([Bru93]) is a navigational way of query formulation in a stratified architecture based on index expressions ([BW92]). Navigational networks for index expressions allow navigation over linguistically motivated subexpression links. Although the INN is developed for the WWW, our approach is mutatis mutandis applicable to all (dynamic or static) information spaces. The required changes only involve the communication of the INN system with the search facilities used to access the information space.

6.2 Related Approaches

Another system based on the approach taken for the INN is the HyperIndex Browser (HIB) ([IWW+95]). The author implemented a first version of this system. Reports describing a rather general introduction to the HIB and experimental results on the cognitive load imposed by the HIB ([DMB98]) have appeared. Compared to the HIB, the INN system uses different techniques for constructing the stratified architecture. In particular, the INN offers a broader notion of subexpressions. Furthermore, the INN system makes use of different search engines than the HIB. For example, the INN also serves the Dutch information community by providing access to two Dutch search engines.
6.3. STRATIFIED ARCHITECTURE FOR INDEX EXPRESSIONS

More loosely related approaches include systems for meta searching, systems using statistically computed refinements, and other hypertext systems exploiting subexpression links. Systems for meta searching, like MetaCrawler and MetaSearch, fire off a query to several search engines and combine their results. Although meta search does offer an overview, it lacks abstraction: the results are simply merged, rather than abstracted.

Statistically computed terms for refining a query are offered by, for instance, AltaVista and Excite. Term co-occurrence frequencies are used to produce a set of related terms. The user can click on (groups of) these terms to add them to the query under construction. These query refinements are statistically rather than linguistically motivated.

In addition, many other hypertext systems exist that use subexpression links. Among the most well-known links are hypernyms (e.g. is-a) and meronyms (e.g. part-of). Those semantical relations between descriptors are, in general, generated from knowledge bases. This means that this kind of semantical approach is domain dependent.

In [VM99], the Condorcet Query Engine (CQE) is presented, a query engine for coordinated index terms. Like the INN, CQE uses structured descriptors. Differences, however, are prominent. For instance, the coordinated concepts of the CQE reside in an ontology. The CQE is thus domain specific and requires careful maintenance of the used knowledge base. In addition, no nested coordinated concepts are allowed in CQE. Therefore, we claim that step-wise refined descriptors are better supported by the INN. Since the CQE is only available as a prototype with a restricted example document space, a pragmatic comparison with the INN is not yet feasible. Whereas the CQE seems suitable for restricted domains and expert users, offering a well-motivated approach, the INN system provides a general dynamic interface for the WWW.

Many stratified architectures, synthesising information retrieval and hypertext, have been proposed. A good introduction and overview is provided in [AS96]. Our approach is based on the stratified architecture for index expressions, which is described in the next section.

### 6.3 Stratified Architecture for Index Expressions

The stratified architecture for index expressions, as briefly described in Section 4.2, augments a set of documents with an ancillary structure, called the hyperindex. This hyperindex forms a abstract description of the contents of the documents. It provides a conceptual overview of the information carried in the documents. The special form of our hyperindex, i.e. a lithoid, which was explained in Section 4.2, allows syntactically enabled navigation. Document exploration is supported by transferring between a point in the hyperindex and actual documents that correspond to a concept in the hyperindex.

The stratified architecture, as depicted in Figure 4.4, consists of two layers, the base layer and the hyperindex, which are connected through the beam relation. The base layer contains the available documents. These documents may be interlinked, for example by hyperlinks. By traversing these links, as is usual in WWW context, navigation in the base layer may
take place. We will not study this further in this thesis. Each document is indexed, yielding a set of descriptors as characterisation. These descriptors reside in the hyperindex.

The beam relation connects the base layer with the hyperindex. That is, the beam relation connects documents in the base layer with the index expressions out of their characterisation. In general, the links between both layers are assumed to be bidirectional, allowing traversal in both directions. From a document, the user may transfer himself to one of the descriptors of the characterisation (beam up). From a descriptor in the hyperindex, the user can perform a beam down, which transfers him to documents that are about the descriptor.

The hyperindex forms an overview of the documents based on their characterisations. The fine-grained structure of lithoids is used in step-wise navigation, as described in the next section.

6.4 Query by Navigation

Formulating an information need aims at finding a descriptor that properly describes it. Query by Navigation (QBN), as briefly discussed in section 4.2, is a navigational way of query formulation in the stratified architecture for index expressions. In this section, we provide a more elaborate description of QBN. By structurally navigating in the hyperindex, users formulate a query. During QBN, documents may be explored by transfers between hyperindex and base layer. QBN identifies two types of actions: navigational actions in the hyperindex (6.4.1) and beam operations for traveling from the hyperindex to the base layer and vice versa (6.4.2).

6.4.1 Navigating the Hyperindex

QBN starts by selecting a single node (index expression) in the hyperindex. The current node in the hyperindex is called the focus. The user may navigate by repeatedly selecting one of the neighbours of the focus. The selected neighbour then becomes the focus and the selection process is repeated. Thus, navigation essentially is a repetitive selection of neighbours. Navigation ends when a satisfactory index expression is reached. Or, when documents that satisfy the information need have been found.

QBN exploits the special form of the hyperindex, i.e. a lithoid, by allowing fine-grained navigation steps. In Figure 6.1, an abstract picture of a lithoid is given in which one of the nodes is marked as focus. The neighbours of the focus depict the direct choices for QBN. They thus give an overview of the concepts available for selection.

In a lithoid, the neighbours of a node are either refinements, direct superexpressions, or enlargements, direct subexpressions. Refinements, residing directly above the focus in the lithoid, denote more specific concepts. For example, conference on IT in (Belgium) is a refinement of both conference on IT and conference in Belgium. Since refinements contain
one node more than the focus, they denote the smallest possible more specific concepts. This guarantees the fine-grained nature of the navigation steps in QBN. By selecting a refinement, the user formulates his need more concisely.

Enlargements denote less specific concepts than the focus. In fact, they denote a subconcept of the focus. For example, IT is an enlargement of conference on IT, which, in turn, is an enlargement of conference on (IT) in (Belgium). By selecting an enlargement, the user thus obtains a broader description. Enlargements can, for instance, be selected in order to recover from a previously selected refinement in order to change direction in the hyperindex. In this way, shifts in interests are fluently dealt with.

QBN is a syntactically enabled navigation mechanism. In [Gro00], it is shown how semantical issues can be employed in QBN. There, it is spelled out how a formal conceptual lattice can be derived from the stratified architecture for index expressions. These concept lattices, containing formally derived concepts based on document characterisations, can reduce redundancy which may enable more direct navigation. Current research should clarify if this approach can be properly applied to dynamic information environments.

In [BB96] it is shown how navigation in the hyperindex can be personalised. Based on information about user interests, additional links are inferred. In [BB97] it is show how the hyperindex can be augmented with semantical information.
6.4.2 Beaming between Layers

Navigation in the hyperindex consists of repeatedly selecting refinements and enlargements. In addition, the user can transfer himself from the hyperindex to the base layer to inspect actual documents. If required, the user can subsequently transfer himself back to the hyperindex, where navigation can be resumed.

Inspecting actual documents is important for a user to ascertain the relevance of documents (searching) and for concept learning (exploration). Inspecting documents is enabled by an operation, *beam down*, that transfers the user from hyperindex to hyperbase (see Figure 4.4). By traveling the beam relation downward, the user is presented with the document that correspond to the focus in the hyperindex. That is, documents that contain the focus in their characterisation. The user can inspect these documents, and, if links between documents are available in the hyperbase, browse through them. If satisfied, the user can end the QBN session. In searching, for example, this may be the case if the user has satisfied his information need by rendering some relevant documents. In exploration, this may happen if the user estimates his knowledge of the field of interest is now sufficient.

In addition, the user may transfer himself back to the hyperindex (*beam up*) to resume navigation. However, since the characterisation of a document may contain several index expressions, a document can be linked to several nodes in the hyperindex. Therefore, the target node for the beam up may not be uniquely defined. This problem, called ambiguity in [Bru93], may lead to user disorientation, especially when the user first follows a few browsing steps. In addition, the INN does not aim at storing the complete stratified architecture. Rather, it generates the required part on the fly. This complicates the computation of beam ups since destinations may be unknown or incomplete. Therefore, the INN system provides a rather basic back-button which guarantees that the user returns in the hyperindex at the same node where he left it.

6.5 INN System

The INN system forms an intermediary between a user and the WWW. It makes use of existing search engines to access information on the WWW. The overall architecture of the INN system is sketched in Figure 6.2. Currently, the INN supports a single user in navigational query formulation. Several search engines that index the WWW are contacted. This allows for different views on the informational content of the WWW. For example, two Dutch search engines are included, specifically adding many national pages.

The path of control of the INN system is illustrated in Figure 6.3. After the user has formulated an initial query, this is translated into the query language of the selected search engine. From the documents returned by the search engine, the titles are stripped. These titles are then parsed, so that index expressions are obtained. Refinements and enlargements in the parsed titles are computed and presented as an overview in a HTML page.
If the user selects one of the navigational options, a refinement or an enlargement, the process is repeated with a new focus. The user may also go to the documents about one of the presented topics (beam down). Those documents then constitute the result of the system.

By providing an example session, we will discuss the workings of the INN system in more detail.

### 6.5.1 Getting Started

The initial screen of the INN system is given in Figure 6.4. The components of this screen are explained below.

The About INN button provides information about the background and workings of the INN system. For example, the idea of QBN, using refinements and enlargements, is explained. The INN Home button leads the user to the home page of the INN system.

To start navigating the WWW, four steps have to be followed by the user.
CHAPTER 6. INDEX NAVIGATOR

Select Search Engine. First, a search engine has to be selected from a list. In Figure 6.4, two well-known search engines, AltaVista and Lycos, are available as well as two Dutch engines, ZOEK and ILSE.

Other search engines can easily be incorporated by slight changes in the communication between the INN system and the search facility. This also means that different information spaces can be navigated and explored via the INN system. Figure 6.2 shows that the communication between the INN system and the information space is mediated by a search facility. This suggests that only changes are needed in the communication between the INN system and this search facility. The user query needs to be translated into a form that the search facility supports. For textual information, this is mostly keyword-based. Since this is already supported, it means that the query translation part need not be changed. The only part that might need changes then is the title extractor. Since most search facilities clearly mark titles in their output, modifying the title extractor is a rather easy task.

Set Size of Result Set. Second, the size of the result set produced by the search facility must be set. That is, the number of documents that will be used for producing
the overview should be specified. In this way, the user can steer the coverage of the overview. In addition, the user gains (some) control over the response time of the system.

The size of the result set can be adapted for each step during navigation. This is a nice property since a larger query generally means that less documents are returned. By increasing the size of the result set, the chances that suitable refinements can be generated also increase.

3. **Provide Initial Query.** Third, an initial query has to be provided. It is interpreted as index expression. The initial query may be of any size. However, it is recommended that a small initial query is provided in order to start with a broad overview.

4. **Send Request.** Finally, the user sends his request by clicking on the send button.

The query is fed to a search engine and the resulting documents are processed. In our example session, the user types the query *retrieval*.

### 6.5.2 Navigating on the Fly

The next page shown to the user, see Figure 6.5, gives an overview of the navigational options. It consists of four parts, which are generated on the fly.

**Focus.** The previously given query, i.e. *retrieval*, functions as current point in the hypertext (focus). By clicking on the magnifying glass next to the focus, the user goes directly to the relevant documents (beam down). This option is offered for all descriptors.

**Refinements.** Next, all refinements of the focus are given. For the example query, refinements include *data retrieval*, *information retrieval*, *retrieval links*, and *storage and retrieval*. The refinement give an overview of the topic *retrieval*.

Refinements are computed within the titles of the documents that were returned by the search facility. Every direct superexpression of the focus that is contained in any of the titles is listed.

The refinements of the focus in another index expression (title) thus need to be computed. This is done by using the twigs of an index expression ([Ude99]). Refinements can only be made if the focus is contained in the title. This is the case if the set of twigs of the focus forms a subset of the twigs of the title.

**Enlargements.** The enlargements of the focus are computed by defoliation (see Section 4.3). An enlargement of an index expression is obtained by removing a single leaf and its connector. In addition, if the root of the focus has only one subexpression, the root (and its connector) can also be removed to obtain an enlargement.
The empty index expression is not included in the INN system since it bears no information. This means that single terms have no enlargements in the INN system. Since the example query is a single term, Figure 6.5 contains no enlargements.

**Related topics.** Some of the titles do not (literally) contain the focus and thus do not lead to refinements. However, since they were rendered by the search facility, they may very well be relevant to the user. Therefore, the top 10 (according to the search engine’s relevance estimates) documents are included. By clicking on the title, direct access to the document is provided. In addition, the magnifying glass is also given which uses the search facility to render documents that are related (about) the title. The selected search engine is used for this.
6.6. EXPERIENCES WITH THE INN SYSTEM

6.5.3 Beaming between Layers

When a satisfactory descriptor has been reached or when an unknown concept is arrived at, the user can transfer his attention to the documents that are about that descriptor (beam down). In the interface, magnifying glasses represent this option. In this way, the user is able to see whether the documents satisfy his information need and he can learn what the concept is about. The back button of the browser enables the user to beam up again.

In the presentation of the documents, the result of the search engine is included as a frame. In this frame, the user can use the facilities offered by the search engine selected. In addition, the user is enabled to return to the INN homepage to start a new navigation session.

6.6 Experiences with the INN System

The INN is a prototype which shows that QBN can be applied to highly dynamic information spaces. This makes it possible to devise formulation tools for BIEs for dynamic environments as well. In particular, the INN could be used as the navigational component of the constructor tool described in Section 5.2.3. For this, the INN has to be integrated with a construction mechanism for BIEs and retrieval functionality to render documents.

Practical experiences with the HIB, a highly similar system, are reported in [DMB98]. The results are promising in terms of cognitive load during query formulation. Additional experiences with the INN are reported in this section.

6.6.1 When No Relevant Documents are Available

For most topics, queries that contain six or more terms do not provide refinements. These queries are too specific. Of course, at a certain point, no new documents are available. However, this point can be delayed by including more index expressions in the characterisations. This means that not only the titles but also (a part of) the contents of documents should be parsed to index expressions. Since most titles are index expressions, we were able to use a simple parser. A more elaborate discussion of extracting index expressions from text was provided in Section 4.7.2.

6.6.2 Dependency on Search Engines

The INN contacts the information space through several existing search engines. This means that the INN depends on the search engines in several ways. For instance, search engines for the WWW tend to change the layout of their pages and interface regularly. This introduces the need to monitor the search engines for modifications and adapt the
preprocessor of the INN if required. Further research should investigate how automatic detection of modifications is best done.

Another dependency on search engines is their incomplete coverage of the WWW. Consequently, the INN suffers from the same problem. It was beyond the scope of our work to address this issue seriously. Further research should examine the integration of several search engines in order to increase coverage.

6.7 Association Index Architecture

The INN mediates between a single user and several sources. Augmentations to this simple setup may involve multiple users or cooperation between several instances of the INN (see Figure 6.6). This section describes our research into the feasibility of such an extension.

6.7.1 Architecture

Information brokers can exchange information between user agents and resource agents. In order for broker agents to be effective intermediaries, they should have knowledge about both users and sources. In the INN, knowledge about documents and their content is modeled by a 2-level hypermedia architecture for index expressions. For filtering, a similar approach can be taken by modelling users and their interest by the 2-level hypermedia for index expressions. In the resulting setup, broker agents have access to two 2-level hypermedia representations. As illustrated in Figure 6.7, brokers then form the connection between the hypermedia representations in the so called association index architecture (AIA).

In [WBW98a], a set of broker agents is considered that cooperatively constitute the association index architecture. Each broker may have its own view on the world. This allows for different ways of matching and specialised brokers. By combining their knowledge about users and sources, the brokers form a distributed AIA.
By mediation, brokers perform filtering and retrieval tasks. When performing a specific task, broker agents will generally use only a part of their knowledge. For instance, when processing a user query, the broker may decide not to use knowledge about other users. To model the selection and use of only the required knowledge, the focus of a broker is introduced. Here, the focus of a broker basically is a subset of his total knowledge. When navigating or browsing in a 2-level hypermedia, one can reside either in the hyperbase or in the hyperindex. Therefore, the focus of a broker is part of the layer it currently resides in.

Broker tasks are implemented as series of actions in the AIA. The actions a broker can perform within the AIA are navigation primitives and transitions. Navigation primitives effectuate a shift in focus within the same layer. Transitions model the migration between layers, delivering a focus in another layer. An example transition comprises the migration from a set of users in the user base layer towards their common interests in the user hyperindex. Transitions are more general than beaming operations as described in Section 4.2.

Figure 6.8 shows several implementations of the transition from lithoid to base layer. In Figure 6.8(a), an embedment transition is provided. Embedment reaches entities in the base layer whose characterisation contains the current focus. As Figure 6.8(b) shows, the about transition yields entities in the base layer whose characterisation is contained in the
current focus. Finally, as sketched in Figure 6.8(c), overlap transitions yield entities that have overlapping descriptors with the focus.

The different forms of transition allow for variation in task implementations. In query processing, for instance, the choice of transition influences the (size of the) resulting set of documents. Additional transitions may be introduced. For example, similarity measures can be applied to refine the beam relation.

6.7.2 Applications of the AIA

Filtering and retrieval tasks in the AIA involve transitions between layers. Consider, for instance, the delivery of a set of documents to interested users. Starting in the base layer for documents, a transition to the hyperindex for documents leads to their characterisations. Then, the broker migrates to the hyperindex for users by mapping the document characterisations to user interests. Finally, a transition towards the base layer for users yields the interested users to whom the documents can be delivered. A similar approach can be taken for retrieval.

To migrate between two hyperindices, the broker may exploit a so called association index (see [WBW98a, WBW98b]). An association index reflects the overlap between two lithoids and consists of nodes and links. The nodes are pairs of index expressions, containing elements from both lithoids to be associated. The links, which are based on the subexpression relation for index expressions, allow navigation towards related nodes.

Since an association index contains information about both users and sources, it may be exploited for matching purposes. To this end, [WBW98a] describes how using a similarity measure for index expressions (see Section 4.6) may be applied to identify the fraction of the association index that is relevant to a specific information need. The article also describes how query expansion and generation may use association indices. Furthermore, it describes how different transitions influence the results of query processing.

In [Mun99], it is described how concept inference ([LZ93]) can be applied in the association index architecture. Concept inference is applied to exploit interests shared by several users during retrieval and filtering. In addition, it is used to model the influence of QBN search paths and lexico-semantical relations in query processing. Initial experiments with a prototype system are promising. For instance, they show that mutual interests can be effectively used to offer more complete results. In addition, they indicate that the final node of a QBN search path is in most cases the best information to use. Adding previous steps of the search path does not yield significantly more relevant documents. However, the small scale test data does not allow a generalisation towards more substantial amounts of data. Further research will be necessary to investigate the applicability of the ideas on a larger scale.
6.8 Outlook

The INN seems a valuable tool for searching and exploring the WWW. Several augmentations would help to improve the usability of the INN. In order to augment the coverage of the INN, additional search engines can be included and their results can be merged. To improve the quality of the search results, numerical matching functions can be applied to re-rank the documents obtained by beaming down.

A graphical interface may enhance the appeal of QBN and may offer a better structured interface. It may, for instance, visualize the direct environment of the focus in the network. Furthermore, it will be interesting to augment the INN with information about user interests. The AIA and the technique of concept inference may provide a good basis for this.

Automatic monitoring of search engines for modifications requires a model of their output. This may be formulated in one of several information modelling techniques such as PSM ([HW93]), NIAM ([NH89]), or EER ([HE92]). Additional research should clarify which techniques are more appropriate.
In order to summarise the results of this thesis, we focus on the research questions discussed in Section 1.3.1. Directions for further research were indicated at the end of each section. Here, we highlight some major issues and provide additional directions.

**Synthesising Retrieval and Filtering** Chapter 2 elaborated on information discovery, the synthesis of information retrieval and filtering. We provided an introduction to information retrieval, including a brief historic overview and a description of several models. In addition, we compared the paradigms of information retrieval and filtering. We identified several benefits of the synthesis of information retrieval and filtering concerning query expansion exploiting user models and profile construction with queries. Further research should aim at capitalising on these issues.

As a major point of concern, we investigated the influence of the duality inherent in information discovery on the design of information brokers by using so called cumulative duality matrices. Further research can aim at the integration of these matrices in agent-oriented design methods.

We described how information discovery can be seen as a multi-agent paradigm. Actual implementations of information discovery may require a refinement of this architecture, for instance exploiting sub-agents to divide and conquer specific functionality. An example of this was presented in Chapter 3.

**Profile Architecture** In Chapter 3, we provided a conceptual overall design for the PROFILE system. This was refined and implemented in the PROFILE prototype. The prototype integrates research done in the four PROFILE components. The suitability of an agent-based architecture for the PROFILE project was evaluated by first
investigating functional and organisational requirements. We then illustrated which properties of agents are useful in addressing the constraints on PROFILE.

Further research can result in instantiations of the PROFILE modules, adding new or improved functionality. The agents in the PROFILE system should then be able to dynamically select the most appropriate partners for cooperation. This necessitates the need for negotiation strategies (e.g. [NPLL96, SV97]).

**Foundation of Index Expressions** A solid formal basis for index expressions was provided in Chapter 4. We spelled out the structural representation of index expressions, exploiting the nesting operator to form the structure of index expressions. This representation was proven equivalent to the grammar and broaden-based representations. The defoliation operator we specified was used to define several notions of subexpressions.

Future research can build upon the formal basis of index expressions. Chapter 5 provided an augmentation of index expressions with logical operators. Other augmentations are possible, such as the explicit inclusion of adjectives. The ideas described in [Ber98] can serve as a starting point. The formal basis of index expressions can also be exploited for the construction of structured navigation layers. For instance, the association index architecture ([WBW98a, WBW98b]) combines lithoids to merge information about users and documents. A conceptual overview of the association index architecture was given in Section 6.7.

**Similarity between Index Expressions** The second main topic of Chapter 4 was the design of numerical similarity functions for index expressions. We first formalised properties concerning the refinement structure of index expressions. We then devised similarity functions that adhered to these properties while abstracting from the actual comparison of keywords. We provided an evaluation in the light of subexpressions. A comparison with simple bag-of-words similarity measures was also provided. In addition, we included “bag-of-twigs” measures in the comparison. The experimental evaluation illustrated the suitability of the designed similarity measures for discriminating between different types of subexpressions.

The devised similarity measures can be incorporated in information systems that exploit index expressions. For instance, the INN system, as described in Chapter 6, might be adapted with these measures to enlarge its discriminating power between documents. Evaluating such augmented systems may provide additional insight in the suitability of the measures. In addition, further research could compare the similarity measures by recall/precision graphs. This would allow their retrieval effectiveness to be compared with other descriptor languages and similarity measures.

**Compact Descriptors** Chapter 5 focussed on the design of a compact descriptor language that finds a balance between expressiveness and tractability. By augmenting index expressions with nested Boolean operators, the language of Boolean index expressions (BIEs) was obtained. Their expressiveness and tractability reside in between that
of index expressions and noun phrases. BIEs enable a compact representation of information needs. The compactness of BIEs was studied, revealing, for example, bounds on compactness and many qualitative properties.

In order to assist users during (re)formulation of their information needs with BIEs, a constructor and modification tool (e.g. as described in Section 5.2.3) should be implemented. Special attention should be paid to different ways of presenting BIEs. For example, textual representation as well as graphical visualisation (see e.g. [HMM99]) and modification should be supported to enlarge the user friendliness of the system. Navigational formulation for index expressions has proven of value for information retrieval. In our opinion, it deserves additional research to investigate navigational formulation for BIEs. Issues of concern are the characterisation of documents in the form of BIEs and the construction of a suitable navigation network.

For matching purposes, it will be interesting to incorporate weights in the similarity scheme for BIEs. To this end, the similarity functions for index expressions should be augmented with weights. Weights can be assigned to complete BIEs, parts of BIEs, single terms and connectors, and to their Boolean operators. For the latter, the Extended Boolean model ([SFW83]) can serve as a starting point. It will also be interesting to look into matching functions that do not require zipping of BIEs. Points of departure for this issue can be found in conceptual graphs ([Sow84, HOC96]).

Advances in natural language processing can yield more complete document characterisations. In [Olt00], for example, a technique for resolving (some) linguistic conjunctions is described. Although this technique certainly does not cover the full range in which conjunctions can be formulated, its robustness contributes to better indexing. Better document indexing eases the path for constructing stratified architectures for BIEs.

Finally, evaluating the suitability of BIEs for information discovery from different points of view is worthwhile. For instance, the use of BIEs may be studied from a cognitive perspective. This should investigate the cognitive load put on users while working with compact descriptors. For index expressions, such a study is reported in [DMB98]. It is worthwhile to evaluate if this approach can also be applied to BIEs. Furthermore, the effectiveness of BIEs in information discovery can be looked into via experiments aiming at recall/precision graphs.

Reducing Syntactical Variety Chapter 5 also provided a way to deal with the syntactical variety in BIEs resulting from the incorporation of Boolean operators. Normalisation was done by zipping up the dyadic operators in BIEs. This results in BIEs in disjunctive normal form, i.e. a logical sum of products of elementary (atomic) BIEs. We showed that this form enables BIEs to be matched with (sets of) index expressions. This enables information needs specified as BIEs to be used for information discovery when documents are characterised by index expressions.

Further research can aim at further evaluation of the suitability of zipping. For instance, one can compare zipping with other methods to normalise BIEs. For this, the
priorities of the dyadic operators can be reversed. In addition, a “left-to-right” priority scheme can be adopted. Bringing BIEs in their most compact form is beneficial for storage and communication. However, this might be a difficult task, considering that bringing propositional formulae into such form is an NP-complete problem ([Weg87]). Additional research should indicate under which circumstances it is feasible to transform BIEs into compact form.

Navigational Query Construction in Dynamic Environments Chapter 6 showed how navigational query formulation in the stratified architecture can be applied to the WWW. By generating only the required part of the architecture on the fly, the size and dynamics of the WWW were coped with. The idea was implemented in the INdex Navigator (INN) which is available via the WWW. The INN exploits existing search engines, thus inheriting their coverage.

Placing the topic of this thesis in the context of the internet and electronic commerce, the author envisages the following. In order to tame the still increasing streams of information, information discovery systems will need to exploit more complete descriptions of information needs. To facilitate the formulation of information needs, retrieval and filtering will be synthesized in a single interface to integrate user-computer interaction for these tasks. Elaborate user profiles will be maintained and updated regularly. Therefore, expressive yet tractable descriptors, such as BIEs, will be needed to represent complex information needs.

Numerous agents, coping with the dynamic nature of information environments, will cooperatively facilitate the exchange of information. In an electronic information market, agents with different tasks will negotiate to establish and revise fruitful cooperations. An integral part of the communicated messages between agents will be the representation of the information needs at hand. Minimising the size of messages must be aimed at to limit the required bandwidth. This is increasingly important in dealing with the growing amount of traffic on the internet. In addition, its importance is stressed further by the advent of wireless networks that feature narrow bandwidth communication channels. This calls for compact descriptors, such as BIEs, which offer the opportunity to convey complex information needs in reasonably sized messages.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


Index

$[I]$, 138
$\Delta$, 73
$\prec_{d}$, 84
$\preceq$, 86
$\text{sim}_{C}$, 90
$\text{sim}_{T}$, 89
$L^{B}_{(T,C)}$, 112
$L^{E}_{(T,C)}$, 70
$L^{\bigstar}_{(T,C)}$, 69
$\circlearrowright$, 69
$\epsilon$, 65
$\equiv$, 130
$\preccurlyeq$, 87
$\otimes$, 64, 68
$\prec$, 85
$\leadsto$, 130
\textbf{Profile}, 13, 47
prototype, 58
requirements, 48
\textbf{AndCons}, 126
\textbf{Atomic}, 115
\textbf{ConCount}, 127
\textbf{ConFree}, 127
\textbf{Conns}, 83
\textbf{DisCount}, 127
\textbf{DisFree}, 127
\textbf{EC}, 94
\textbf{Exp}, 139
\textbf{FP}, 93
\textbf{HasComp}, 127
\textbf{IsMaxPos}, 153
\textbf{IsMin}, 144
\textbf{MaxExp}, 142
\textbf{MinExp}, 141
\textbf{NegMax}, 162
\textbf{NotProd}, 125
\textbf{OrCons}, 124
\textbf{OrProd}, 125
\textbf{Terms}, 83
\textbf{ZipAnd}, 126
\textbf{ZipOr}, 124
\textbf{Zip}, 124
\textbf{add}, 81, 112
\textbf{compact}, 142
\textbf{head}, 83
\textbf{EqOrder}, 91
agents
\textbf{Profile}, 58
broker,
see information, broker 36
middle, 37
query formulation, 52
resource, 36
user, 35, 57
\textbf{AIA}, see association index architecture
188
association index architecture, 188
beam
down, 66, 182
up, 66, 182
\textbf{BIE}, 111
atomic, 115
characterisation, 169
compiler tool, 116
disjunctive normal form, 123
implementation, 172
matching, 169
maximal, 160
maximal positive, 153
representable, 154
minimal, 144
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority scheme</td>
<td>120</td>
</tr>
<tr>
<td>propositional form</td>
<td>123</td>
</tr>
<tr>
<td>umbrella</td>
<td>158</td>
</tr>
<tr>
<td>negated</td>
<td>164</td>
</tr>
<tr>
<td>zipping</td>
<td>120</td>
</tr>
<tr>
<td>conjunctions</td>
<td>126</td>
</tr>
<tr>
<td>disjunctions</td>
<td>124</td>
</tr>
<tr>
<td>function</td>
<td>123</td>
</tr>
<tr>
<td>result</td>
<td>126</td>
</tr>
<tr>
<td>Boolean index expression, see BIE</td>
<td></td>
</tr>
<tr>
<td>broaden</td>
<td>81</td>
</tr>
<tr>
<td>characterising</td>
<td>20</td>
</tr>
<tr>
<td>closed world assumption</td>
<td>115, 125</td>
</tr>
<tr>
<td>compactness</td>
<td>12, 13, 114, 118</td>
</tr>
<tr>
<td>congruence</td>
<td></td>
</tr>
<tr>
<td>dyadic</td>
<td>130</td>
</tr>
<tr>
<td>phased</td>
<td>131</td>
</tr>
<tr>
<td>connector</td>
<td>63</td>
</tr>
<tr>
<td>descriptor language</td>
<td>11</td>
</tr>
<tr>
<td>document</td>
<td></td>
</tr>
<tr>
<td>collection</td>
<td>20</td>
</tr>
<tr>
<td>stream</td>
<td>31</td>
</tr>
<tr>
<td>duality</td>
<td>38</td>
</tr>
<tr>
<td>cumulative</td>
<td>38</td>
</tr>
<tr>
<td>matrix</td>
<td>39</td>
</tr>
<tr>
<td>dyadic count</td>
<td>138</td>
</tr>
<tr>
<td>enlargement</td>
<td>66, 180</td>
</tr>
<tr>
<td>equivalence</td>
<td>130</td>
</tr>
<tr>
<td>expansion</td>
<td>139</td>
</tr>
<tr>
<td>maximal</td>
<td>140</td>
</tr>
<tr>
<td>measure</td>
<td>137, 143</td>
</tr>
<tr>
<td>minimal</td>
<td>140</td>
</tr>
<tr>
<td>expressiveness</td>
<td>11, 12, 114</td>
</tr>
<tr>
<td>headedness</td>
<td>55, 91</td>
</tr>
<tr>
<td>empty</td>
<td>65</td>
</tr>
<tr>
<td>language</td>
<td>70</td>
</tr>
<tr>
<td>lithoid</td>
<td>65</td>
</tr>
<tr>
<td>matching</td>
<td></td>
</tr>
<tr>
<td>criteria</td>
<td>89</td>
</tr>
<tr>
<td>evaluation</td>
<td>97</td>
</tr>
<tr>
<td>function</td>
<td>91</td>
</tr>
<tr>
<td>nesting operator</td>
<td>68</td>
</tr>
<tr>
<td>query by navigation</td>
<td>66</td>
</tr>
<tr>
<td>representation</td>
<td></td>
</tr>
<tr>
<td>broaden-based</td>
<td>81</td>
</tr>
<tr>
<td>grammar</td>
<td>79</td>
</tr>
<tr>
<td>structural</td>
<td>68, 78</td>
</tr>
<tr>
<td>size</td>
<td>70</td>
</tr>
<tr>
<td>stratified architecture</td>
<td>66, 179</td>
</tr>
<tr>
<td>subexpression</td>
<td>83</td>
</tr>
<tr>
<td>direct</td>
<td>84</td>
</tr>
<tr>
<td>embedded</td>
<td>86</td>
</tr>
<tr>
<td>general</td>
<td>85</td>
</tr>
<tr>
<td>strict</td>
<td>85</td>
</tr>
<tr>
<td>twig</td>
<td>96</td>
</tr>
<tr>
<td>InIndex Navigator</td>
<td>182</td>
</tr>
<tr>
<td>indexing</td>
<td>20, 53</td>
</tr>
<tr>
<td>automatic</td>
<td>22</td>
</tr>
<tr>
<td>information</td>
<td>20</td>
</tr>
<tr>
<td>agents</td>
<td>35</td>
</tr>
<tr>
<td>broker</td>
<td>36</td>
</tr>
<tr>
<td>design</td>
<td>38</td>
</tr>
<tr>
<td>types</td>
<td>37</td>
</tr>
<tr>
<td>discovery, see information discovery</td>
<td></td>
</tr>
<tr>
<td>filtering, see information filtering</td>
<td></td>
</tr>
<tr>
<td>need</td>
<td></td>
</tr>
<tr>
<td>dynamic</td>
<td>42</td>
</tr>
<tr>
<td>persistent</td>
<td>31</td>
</tr>
<tr>
<td>privacy</td>
<td>41</td>
</tr>
<tr>
<td>retrieval, see information retrieval</td>
<td></td>
</tr>
<tr>
<td>information discovery</td>
<td>33</td>
</tr>
<tr>
<td>agent-based</td>
<td>34</td>
</tr>
<tr>
<td>information filtering</td>
<td>11, 30</td>
</tr>
<tr>
<td>collaborative</td>
<td>32</td>
</tr>
<tr>
<td>information need</td>
<td>20</td>
</tr>
<tr>
<td>information retrieval</td>
<td>11, 19</td>
</tr>
<tr>
<td>history</td>
<td>21</td>
</tr>
</tbody>
</table>
models, 23
  boolean, 23
  cognitive, 30
  inference network, 28
  logical, 26
  natural language, 29
  probabilistic, 25
  vector space, 24
INN, see Index Navigator
IR, see information retrieval
lithoid, 180
nesting operator, 64, 68
normalisation
  linguistic, 54
noun phrase, 54, 105
pointing sequence, 71
precision, 22
preferential model, 27, 67
query, 20
  expansion, 52
  formulation, 178
query by navigation, 180
  on the fly, 185
recall, 22
refinement, 66, 180
relevance feedback, 20, 117
SIGIR, 22
tractability, 11, 12, 114
TREC, 23
twig, 88
user
  interface, 57
  modeling, 51
World Wide Web, 15, 47, 177
Samenvatting

Met deze samenvatting wil ik een indruk geven van het onderwerp en de resultaten van dit proefschrift. De hoofdstukindeling vormt daarvoor de kapstok.

1. Introduction De hoeveelheid electronisch beschikbare informatie heeft de laatste jaren een vogelvlucht genomen. Eén van de belangrijkste redenen hiervoor is de enorm gestegen populariteit van het World Wide Web (WWW). Het zoeken naar electronisch opgeslagen informatie is voor menig een vaak voorkomende activiteit. Bij het zoekproces worden beschrijvingen gebruikt om de informatiebehoeften van gebruikers en de inhoud van documenten weer te geven. Deze beschrijvingen, die descriptoren genoemd worden, worden ook gebruikt om informatiebehoeften met de inhoud van documenten te vergelijken. Er worden descriptoren met verschillende eigenschappen toegepast in moderne ontsluitingssystemen. In dit proefschrift worden drie criteria beschreven waarmee descriptoren vergeleken kunnen worden: expressiviteit, berekenbaarheid (tractability) en compactheid. Dit proefschrift beschrijft tevens een nieuwe beschrijvingstaal die een compacte representatie van complexe informatiebehoeften toestaat en een werkbare balans vindt tussen expressiviteit en berekenbaarheid.


4. Index Expressions Index expressies vormen een beschrijvingstaal die gebaseerd is op termen en connectoren. Termen geven concepten weer en connectoren worden gebruikt om relaties tussen concepten uit te drukken. De structuur van index expressies kan geïnterpreteerd worden als concept-modificatie. Dit wil zeggen dat de hoofdterm het belangrijkste concept weergeeft en onderliggende, via connectoren verbonden, concepten het hoofdconcept specifieker maken. Ontbladering van index expressies wordt uitgewerkt. Ontbladering is een belangrijk mechanisme waarmee subexpressies berekend kunnen worden. Tevens worden drie verschillende representaties van index expressies vergeleken. Tenslotte wordt het vergelijken (matching) van index expressies beschreven. Een aantal similariteitsfuncties die zowel de inhoud als de structuur van index expressies in beschouwing nemen wordt geïntroduceerd.

5. Boolean Index Expressions De nieuwe beschrijvingstaal, Boolean Index Expressies (BIEs), wordt gedefinieerd. BIEs bevatten zowel de concept-modificatie structuur als logische structuur. Deze logische structuur wordt verkregen door concepten te construeren met Booleaanse operatoren voor disjunctie, conjunctie en negatie. Op deze manier wordt een beschrijvingstaal verkregen die expressiever is dan index expressies en toch voldoende berekenbaar. Bovendien staan BIEs het toe om complexe informatiebehoeften compact te representeren. De compactheid van BIEs wordt bestudeerd met behulp van een normalisatie die Booleaanse operatoren ontwikkeld. Dit resulteert in BIEs in disjunctieve normaalvorm die meestal meer termen, connectoren en/of operatoren bevatten dan het origineel. Deze groei wordt expansie genoemd en vormt de basis van de studie naar compactheid van BIEs. De onder- en bovengrens van expansie worden bestudeerd. Dit gebeurt door een kwalitatieve beschrijving van respectievelijk minimaal en maximaal expanderende BIEs. Tevens worden kwantitatieve aspecten van deze BIEs onderzocht. De vergelijking tussen BIEs wordt beschreven middels een similariteitsfunctie die gebruikt maakt van de genoemde normalisatie. Tevens wordt een indruk gegeven van de implementatie van BIEs in een functionele programmeertaal.

6. Index Navigator Bij het formuleren van de informatiebehoeftes speelt een aantal problemen. Voorbeelden hiervan zijn dat de gebruiker zijn informatiebehoeftes vaak niet concreet kan beschrijven, dat informatiebehoeften kunnen veranderen door interactie met informatiesystemen en dat zoekvragen vaak weinig specifiek zijn. Als mogelijke oplossing is navigerend formuleren voorgesteld, waarbij de gebruiker zich stapsgewijs door een netwerk beweegt. Het netwerk vormt een beschrijving van de (inhoud van) in de collectie aanwezige documenten. Een dergelijk netwerk kan gebaseerd zijn op index expressies. Er wordt beschreven hoe navigerend formuleren met een dergelijk netwerk toegepast kan worden op het WWW. Problemen die hierbij spelen zijn de grootte en de dynamiek van het WWW. Deze problemen worden omzeild door “on the fly” alleen het gewenste deel van het netwerk te genereren. De beschreven aanpak kan mutatis mutandis gebruikt worden voor alle dynamische collecties.
Curriculum Vitæ

Personalia

Name: Bernd Christian Martijn Wondergem
Date of birth: February 5th 1973
Place of birth: Ede (Gld), The Netherlands
Address: University of Nijmegen
  NL-6525 ED Nijmegen
  The Netherlands
Email: bernd@cs.kun.nl

Education

1985-1991 VWO, Christelijk Streeklyceum, Ede
1991-1996 Computer Science, Utrecht University
1996-2000 PhD student, Computer Science, University of Nijmegen
(page deliberately left blank)
(page deliberately left blank)
(page deliberately left blank)
(page deliberately left blank)
(page deliberately left blank)