August 1980

KNOWLEDGE-BASED RETRIEVAL ON A RELATIONAL DATABASE MACHINE

by

David Elliot Shaw

ABSTRACT

The central focus of this research has been the efficient retrieval of records from very large databases in applications where the criteria for description-matching require deductive inference over a domain-specific "knowledge base". Our approach has involved the design of a specialized non-von Neumann machine which permits the highly efficient evaluation of certain operators of a *relational algebra* of particular importance to the computational task of logical satisfaction. The architecture permits an $O(\log n)$ improvement over the best known evaluation methods for these operators on a conventional computer system, and may also offer a significant improvement over the performance of previously implemented or proposed database machines in other applications of practical import.

This thesis was submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2494, Contract MDA903-77-C-0322. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, or any agency of the U. S. Government.

This dissertation is dedicated to

Dr. Charles B. Shaw, Jr.:

father, scientist, friend

ACKNOWLEDGEMENTS

Of the many individuals who have contributed in various ways to the completion of this dissertation, the most difficult to adequately acknowledge is Terry Winograd. Serving not only as thesis advisor, but also as an attentive audience, a consistently constructive critic, a clever broker of ideas, a cheerful administrative manager, a reliable source of encouragement, and on occasion, a gifted therapist, Terry has gone far beyond the usual bounds of supervisory responsibility in his efforts to see this work through to its completion.

Gio Wiederhold has also exerted a strong influence on the content and methodology of the research reported in this thesis, and has played a critical role in the discovery of what now appear to be the most surprising and potentially important results of this research. Without his support, and that of the other members of the Knowledge Base Management Systems Group at Stanford, this work would not have been possible.

Special thanks are due to Don Walker, of SRI International, for generously agreeing to serve as an outside committee member, for his careful reading of this dissertation, and for his kind and thoughtful counsel on several occasions. Discussions with Bob Floyd and Don Knuth at critical points in the course of this research also proved quite valuable, as did the valuable insights provided by Ed Parker over a period of several years.

Among the other individuals who have made significant contributions of various sorts to the completion of this dissertation are Dennis Allison, Cleve Ard, Al Baisuck, Phil Bernstein, Bruce Berra, Alan Borning, Denny Brown, Corky Cartwright, Don Coates, Lew Creary, Dot Dale, David DeWitt, Donald Dunn, Les Earnest, Marc Eisenstadt, Mike Farmwald, Bill Faught, Ed Feigenbaum, Shel Finkelstein, Marty Frost, Dick Gabriel, Ralph Gorin, Cordell Green, Brian Harvey, Matthew Heiler, Gary Hendrix, Leslie Hertzig, Carl Hewitt, Susan Hill, Fred Holtby, David Hsiao, J. Q. Johnson, Jerry Kaplan,

Kevin Karplus, Arthur Keller, Jonathan King, Lori Konovitz, Fred Lakin, Juan Ludlow, Jack MacCrisken, Stu Madnick, Paul Martin, John McCarthy, Margaret Morton, Don Norman, Ted Panofsky, Jayne Pickering, Irving Pfeffer, Rob Poor, Michael Ray, Jeff Rubin, Paul Saltman, Daniel Sagalowicz, Rosemary Seiter, Marilyn Shaw, Charles B. Shaw, Jr., Ken Shoemake, Carolyn Tajnai, Luis Trabb-Pardo, Jeff Ullman, Richard Weyhrauch, Curt Widdoes, Jeanette Wirz, and the entire staff of Stanford Systems Corporation.

The mathematical results presented in Subsection 3.6.2 were derived using the MACSYMA system at MIT, which was developed in part through the support of the United States Energy Research and Development Administration and the National Aeronautics and Space Administration. In this regard, further thanks are due to Don Knuth and Luis Trabb-Pardo for their roles in developing and assisting in the use of the remarkable TEX system for mathematical typesetting, and more generally, to the staff and students responsible for the facilities of the Stanford Artificial Intelligence Laboratory.

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under contract MDA903-77-C-0322.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

Doctoral dissertations are typically characterised by a very detailed treatment of a very narrow topic whose broader significance may be difficult to argue before all but a very specialized audience. It is our feeling, however, that dissertations could not be written if it were not possible for the authors to imagine their work to be part of a much larger framework of broad and globally significant questions. Whether one regards the perceived relationship between a thesis topic and the great unsolved problems of an academic discipline as a reflection of actual insights or as a delusion necessary to motivate completion of the dissertation, it has been our experience that an appreciation of this framework is often quite helpful to the reader in following and integrating the details of the thesis.

We thus begin this dissertation by acknowledging our own particular delusion.

The fundamental problems which we have recently found most exciting are concerned with the relationship between very high level mechanisms for the *description* of computational tasks and highly efficient parallel hardware architectures for their *execution*. On the one hand, we might ask for the ability to describe such tasks in a natural, perspicuous manner, much as they might be described to a human programmer. (The tremendous utility of such a capacity for high-level description is by now quite familiar to most computer scientists; its practical implications in the areas of software expenditures, program reliability, and ease of maintainance and modification will not be belabored here.) Such descriptive mechanisms are of little economic value, though, if they can be interpreted only by processes whose execution requires prohibitive amounts of time. Since the time complexity of an algorithm is inextricably tied to the architecture of the underlying machine, it is natural to be concerned with the interactions between linguistic features and the hardware

substrates on which they will ultimately be realized

In this regard, it has been our suspicion that characteristics not intrinsic to the process of computation, but associated instead with the particular constraints of the von Neumann machine may have made it difficult for computer scientists to *conceive* of tools which might well support major advances in program description. Even if well-founded, this suspicion should not be regarded as a criticism of present designers of programming languages and systems, the quick elimination from consideration of linguistic features whose implementation would clearly be hopelessly inefficient is probably an important heuristic, allowing the designer to prune the enormous tree of possibilities to a manageable size. The rapidly emerging options for extensive *hardware parallelism* within economically feasible systems, however, introduce the possibility that such heuristics may interpose an impassable conceptual block between the designer and an unorthodox, but vastly superior solution to this important problem. In the course of our research, we have thus attempted to free ourselves on a temporary basis from preconceived notions of algorithmic efficiency, endeavoring to consider with minimal prejudice a wide range of mechanisms which might offer great power and generality, and deferring judgments as to their practicality until a variety of architectural options for their efficient implementation have been considered.

There is no sense in which we can claim to have systematically explored any well-defined space of possible descriptive mechanisms or hardware architectures, our explorations in both areas have been strongly and unavoidably influenced by our background, interests and intuitions, as well as by those of our colleagues. It might be appropriate at this point to acknowledge some of these influences, both to prepare the reader for the directions in which our research has proceeded, and to reveal the sources of hidden assumptions, biases

and implicit conceptual metaphors which may surface in the course of our exposition. The themes of nonprocedural description, the use of domain-specific knowledge and the explicit axiomatization of computational semantics could well be regarded by the skeptic as an artifact of our academic environment at Stanford. Similarly, one might expect that our great personal interest in and intellectual affinity for the principles of parallel processing, using such mechanisms as content-addressability, intelligent circulating storage, locally interconnected cellular arrays and other manifestations of extensive distribution of logic throughout large amounts of storage, might well direct our investigations toward the incorporation of such mechanisms within our architectural solution. The effects of both of these influences should be evident throughout this dissertation.

1.1 The General Conceptual Matching Problem

We have thus far attempted only to convey an intuitive feeling for some of the more important contemporary problems of computer science of which our own thesis topic might be considered a small and restricted, but—to the extent that we have been successful—significant and tractable, instance. Let us now move to a somewhat less abstract characterisation of the kind of problem with which our dissertation research has been concerned.

Many information processing tasks performed by men and machines alike involve the process of *matching*, by which a correspondence is assigned between members of two sets of entities. The criteria for certain sorts of matches are quite simple to describe. Letters are routinely matched with mailboxes, for example, and Congressmen with constituents, according to straightforward algorithms based on simple, single properties of the entities in question. By contrast, our doctoral research may be thought of as concerned with a more interesting class of tasks which might be termed *conceptual* matching problems. This more demanding sort of task, which is nonetheless a common part of our cognitive experience, involves the assignment of a correspondence between entities which humans perceive as being highly and systematically structured, based on selected significant characteristics of those structures.

A surprisingly large share of the kinds of information processing activities with which both human and automated data processors are charged may be viewed as involving various kinds of conceptual matching problems. Although meaning-based matching may, in a COBOL-based inventory control system for airplane parts, be deeply embedded within program loops and sort routines, and hence difficult to recognise as such, the fact remains that a large proportion of the CPU cycles which will be expended in any particular run might be

thought of as identifying appropriate records for manipulation on the basis of domain-specific criteria—in our example, criteria involving, say, airplanes, motors and aircraft-specific part-whole relationships. In the general spirit of much of the recent work on knowledge-based systems, we might ask for the ability to describe these matching criteria, along with the domain-specific entities and relationships on which they are based, in a very direct, modular, easily understandable manner, mapping salient facts, rules and relationships onto independently expressable assertions within the programming system.

Previous work in the field of artificial intelligence offers a rich set of knowledge representation and matching techniques which might be employed in the pursuit of this general approach to the problem of conceptual matching. The kinds of applications with which we are most concerned, though, are those in which the amount of data to which conceptual matching techniques must be applied may be quite large. More specifically, our doctoral research attacks the problem of matching a given *pattern* description against the members of what may be a *very large* set of candidate *target* descriptions according to meaning-based criteria. The potential size of the collection of target descriptions imposes special constraints on the sorts of conceptual matching techniques that might be successfully applied in practice.

1.2 Knowledge-Based Information Retrieval

Having wallowed in various levels of abstraction for some pages now, we are prepared to describe the particular concrete example of a large-scale conceptual matching task that we have used to exercise the ideas on which our thesis research is based. This task is borrowed from the general paradigm of *information retrieval*, and is itself most easily exemplified by the *document retrieval* (more accurately, *reference* retrieval) application. In an ordinary document retrieval system, a collection of target *documents*—all the books in a computer science library, for example—is first *indexed* by associating a *target* description with each document in the collection. The end user of the system, who we will call the *searcher*, then prepares a *pattern description* which embodies some of the salient characteristics of the sorts of documents in which he is interested. The system then compares the pattern description with the candidate target descriptions in the collection, returning all targets that "match" according to certain prespecified criteria.

It is the nature of these criteria that distinguishes the behavior of a *knowledge-based* information retrieval system, and indeed, of a system for conceptual matching in general. In such applications, it is not possible in general to decide whether a match should succeed in a strictly mechanical, "syntactic" manner; instead, the acceptability of a match may depend on domain-specific entities and relationships, and on deductive inferences over these entities and relationships. In the case of the computer science library, for example, the system might be required to "know about" such entities as *computers*, *algorithms*, *programmers*, and *storage devices*.

Certain *characteristic attributes* of these entities (the *storage medium* attribute, whose values differ for different kinds of *storage devices*, for example)

might also be included in this domain specific knowledge. Among the typical kinds of relationships that might be embodied in the *knowledge base* of such a system is the fact that a *tape drive* is a particular kind of storage *device* whose *storage medium* is always *magnetic tape*; one simple deductive inference involving this relationship might establish the fact that a pattern description in which the subject of a document is described as involving a *storage device* with *magnetic tape* as its medium would be satisfied by a target description in which a *tape drive* appeared in the corresponding position.

Let us now briefly examine the manner in which such a knowledge-based information retrieval system might be used in practice. In contrast with an ordinary information retrieval system, three distinct classes of users would be involved in the operation of a knowledge-based retrieval system. In addition to the searchers and indexers, a third class of users having expertise in the subject areas of the documents to be indexed would be required to formulate and encode the sorts of domain-specific knowledge described above for use in indexing and retrieval. Members of this third class of users, which has no analogue within the conventional information retrieval system, might be called *knowledge engineers*. Our primary concern in this dissertation, however, will be with the process of retrieval by searching end-users, under the assumption that the knowledge base has previously been constructed and all documents in the collection indexed.

In the next chapter of this dissertation, we will review the typical characteristics of existing information retrieval systems, attempting to identify certain significant weaknesses that are addressed by the knowledge-based approach to retrieval and thus motivate the particular application in which we have embodied our ideas regarding the abstract process of conceptual matching. While it would be premature at this point in our discussion to try to

characterise the sorts of reference retrieval tasks for which our approach would be most appropriate, it is worth mentioning that this approach offers the greatest advantages in the case where highly specific targets are to be retrieved from among a large class of "conceptually heterogeneous" documents. Phrased differently, knowledge-based retrieval methods should prove most critical in the context of tasks in which the semantic criteria for satisfaction of a user's request are meaningful for only a comparatively small subset of the target collection. A very ambitious example of such a task might be the selection of a specialized journal article whose relevance might only be apparent to, say, a graduate student working in the field who had read the paper, from among the set of all documents in a large university collection.

1.3 A System for Knowledge-Based Retrieval

As part of our doctoral research, we have implemented a very simple knowledge-based information retrieval system having the basic structure which we have just outlined. In the interest of applicability to problems other than our sample document retrieval application, however, the system we have implemented is in fact somewhat more general in one respect than suggested by the discussion of Section 1.2. Specifically, the rules defining the semantics of matching within the document description language have not been embedded inextricably within the code of the retrieval system, but have instead been explicitly formulated as an independent, separable set of axioms expressed in restricted first-order predicate calculus. Specification of the match semantics in the form of a separate set of declaratively specified rules also contributes to the flexibility of our thesis system, in that the matching axioms could be easily modified to reflect changes in the description language or in the rules for description matching without affecting the behavior of the system as a whole through modifications of the code itself. More accurately, then, the knowledge-based retrieval system which we have implemented may be thought of as making reference to three conceptually distinct "databases": a target collection, a domain-specific knowledge base, and a match specification defining the matching semantics of the knowledge-based description language.

As we will see in later chapters of this dissertation, the flexible approach to the the specification of match semantics which we have outlined, together with the capacity for the use of domain-specific knowledge in evaluating the success of potential matches, supports a very powerful and highly general set of capabilities which are not available in a conventional information retrieval system. It is not difficult to construct a scenario in which this sort of generalized

knowledge based retrieval system might offer a number of important practical advantages by comparison with a conventional information retrieval system. Unfortunately, there is a fundamental respect in which our presently operational demonstration system, which runs on conventional computer hardware, would not be practical for use in an actual application involving a large target collection.

Specifically, the demonstration system relies very heavily on the execution of several operations which, on a von Neumann machine, are quite expensive when the operands comprise a large amount of data. Since the case of a very large target collection is, for reasons which will become clear shortly, precisely the situation in which our knowledge-based approach is of the greatest potential utility, we have been forced to consider various alternatives to the von Neumann machine architecture which support these operations much more efficiently in order to justify the promise of our approach in practical applications.

The result has been the design of a non-von Neumann machine on which certain operations--in particular, the most computationally expensive primitive operators of a *relational algebra*--may be evaluated in a highly efficient manner. Based on a hierarchy of associative storage devices, this architecture in fact permits an $O(\log n)$ improvement (with very favorable constant factors) in time complexity over the best known evaluation methods for these operators on a conventional computer system, without the use of redundant storage, and using currently available and potentially competitive technology. Although it was not our original goal in pursuing this research, these results have recently begun to attract attention within the database management community by virtue of the important role played by these "difficult" relational algebraic primitives within database management systems based on the relational model of data [Codd, 1970]. It is this connection with the concerns of relational database systems, together with the close relationship between our architectural work and earlier research on specialised hardware for database management, which has motivated our use of the phrase "relational database machine" to describe our proposed architecture.

On close examination, this more immediate byproduct of our research is not quite the coincidence it might seem at first glance; as will be seen in Chapter 2, the reasons these operations have proven to be so central to our own work is closely related to at least one of the reasons for their frequent appearance in the relational database literature. Still, it must be emphasised that the research problem that motivated the design of our machine differs in a number of important respects from most contemporary database management tasks. While the potential applications of this sort of machine to the construction of very high performance database management systems are certainly of interest, a conclusive judgement regarding their immediate practical utility in such applications would consequently be premature at present.

1.4 Organisation of the Dissertation

The nature of the problem we have chosen to attack, and of the approach to its solution that we have adopted, has imposed what might be described as a *vertically integrated* organisation on this dissertation. Although we are concerned with only a single task, which is itself reasonably well defined and manageable in scope, the methodical reader will be following our approach to its solution along a long route extending from the level of very high level descriptions, through the arena of logical formula manipulation, and down to the domain of actual parallel operations in hardware. It has been our experience in describing this research before a number of audiences that the "vertical distance" which must be covered in its exposition may make it difficult to retain the overall structure of our thesis work while attending to the details of each of these "layers".

In an attempt to mitigate this difficulty, we have attempted to illustrate in Figure 1.1 the vertical structure of the sort of system which we propose, which is closely mirrored by the organisation of this dissertation; the reader may wish to refer back to this figure periodically in the course of reading our thesis for purposes of reorientation (or at very least, reassurance). At the top level of our proposed system organization, documents and domain-specific knowledge are represented using a *knowledge-based description language*. Examples of this language appear immediately to the right of the phrase "knowledge-based description language" in Figure 1.1; they need not be studied carefully at this point, but are presented to give some feeling for the kind of information they embody. At the bottom of our vertical chain, the primitive operations of a relational algebra (again illustrated immediately to the right of the corresponding title) are interpreted by our proposed *relational database machine*.

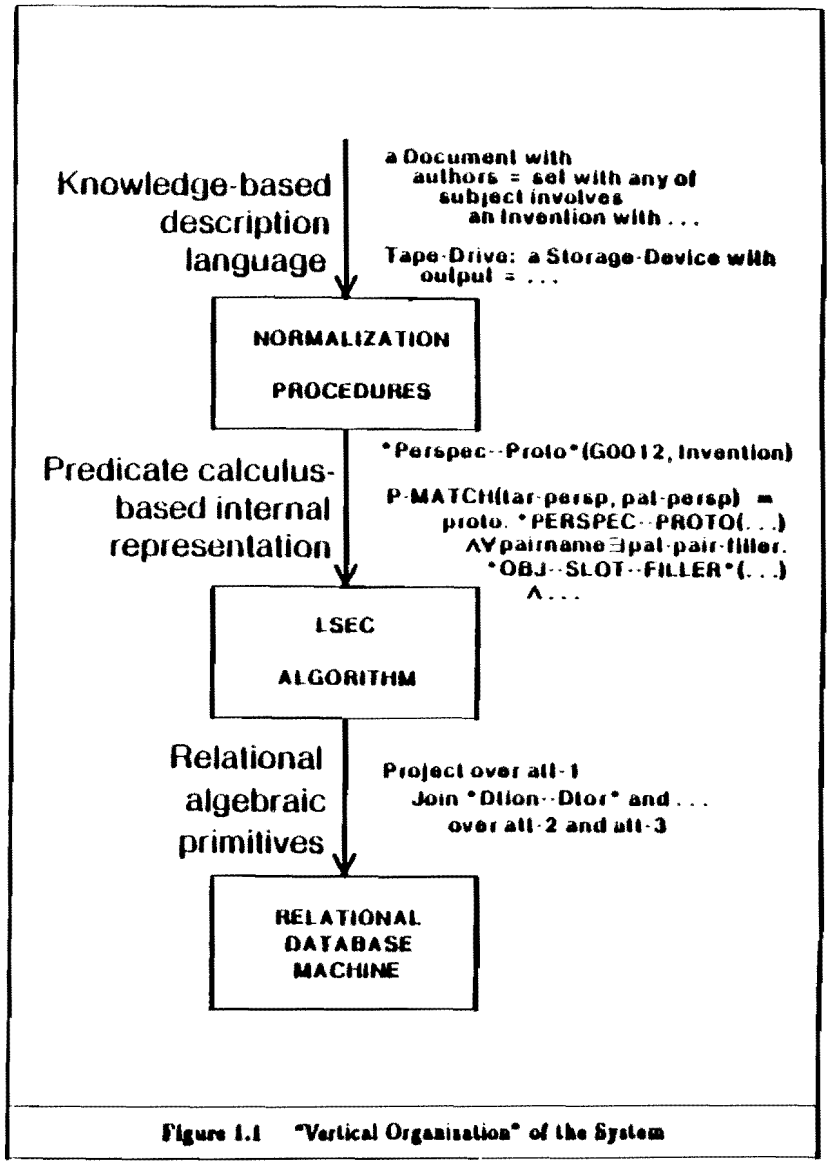

Figure 1.1 "Vertical Organisation" of the System

No hardware was actually built in the course of our doctoral work; the essential behavior of the our proposed machine architecture, however, has been emulated in software as part of our working demonstration system, and the hardware processes that would replace these emulating routines in an actual implementation have been analysed algorithmically in some detail.

Most of the activity of our actual thesis system, however, takes place in the middle portion of our illustration--the part dealing with formulæ expressed in a restricted first order logic. As we shall see, the strength of predicate calculus as an "intermediate form" for our thesis system derives from its flexibility as a descriptive tool, on the one hand, and from its interesting and useful relationship to the relational algebraic primitives, on the other. In our thesis system, all descriptions, both target and pattern, are first converted into a special *normalized* form, and the resulting data structures manipulated according to the rules specified in the match specification (which, we recall, are also expressed in predicate logic) to locate all matching target documents. The details of these manipulations are embodied in an algorithm called *I.SEC* (for *Logical Satisfaction by Extensional Constraint*), which has as its primitive operations the relational algebraic operations that constitute the macro-instruction language for the relational algebraic machine.

As noted above, the organization of this dissertation follows quite closely the structure of the system itself. At the top level, our exposition is divided into two parts: in Chapter 2, we describe the structure and behavior of the running demonstration system, which implements a simple knowledge-based retrieval task; Chapter 3 outlines the architecture of the proposed machine, and specifies the way in which it would actually perform the primitive relational algebraic operations that are emulated in software in the demonstration system.

More specifically, the second chapter reviews the essential characteristics

of conventional information retrieval systems, motivates the formulation of our knowledge-based description language, and describes the manner in which the I.SEC algorithm is used for retrieval based on these descriptions. The third chapter reviews previous work on associative processors and database machines, describes our proposed architecture, and introduces and analyses the complexity of the parallel hardware algorithms used to evaluate each of the relational algebraic primitives on this machine.

CHAPTER TWO

THE KNOWLEDGE-BASED RETRIEVAL SYSTEM

In this chapter, we will motivate and describe the demonstration system which we have implemented in software as part of our doctoral research. The demonstration system actually retrieves matching document descriptions from a small target collection on the basis of simple domain-specific knowledge, emulating the macro-instructions of the proposed database machine in a reasonably straightforward, though not particularly efficient, manner. The system, which was written in MACLISP on the DEC PDP-10 at the Artificial Intelligence Laboratory at Stanford, has been tested using a suite of sixty representative pattern descriptions.

The chapter begins with a survey of conventional information retrieval systems, providing the background for a discussion in Section 2.2 of certain limitations of these systems which are addressed by the concerns of knowledge-based retrieval. The remainder of the chapter describes the structure and function of the demonstration system, from the level of knowledge-based descriptions through the invocation (but not the evaluation) of the relational algebraic primitives which form the macro-instructions of the proposed database machine. Section 2.3 introduces and exemplifies the knowledge-based description language. In Section 2.4, those aspects of the relational model of data, its relationship to predicate calculus, and its use as a database query language that are essential to our own work are presented and illustrated; the application of these techniques in our own work is then introduced. Finally, Section 2.5 describes the LSEC algorithm, whose execution is central to the operation of the working demonstration system.

2.1 Conventional Information Retrieval Systems

This section is intended to provide an overview of previous and ongoing research related to information retrieval systems (more precisely, *reference* retrieval systems) having substantial relevance to the concerns of our thesis research. Generally speaking, we intend the term *reference retrieval system* to denote a system used to store and retrieve *reference* records bearing a one-to-one correspondence to the documents in some collection, and containing some information about the document, including information useful in physically retrieving the document itself. We will be especially, though not exclusively, concerned with *computer-based* systems for reference retrieval.

In the current section, we will outline the essential functional characteristics of typical past and current reference retrieval systems, along with the methods and characteristic results of system evaluation, using a fairly traditional expository approach relying heavily on review of the pertinent literature. In Section 2.2, however, a unified analytical framework will be presented for viewing the characteristics and limitations of such systems, oriented toward the central theoretical concerns of our research. It should also be noted that many aspects of information retrieval having great practical significance (research in automatic indexing and specific approaches to system implementation, for example) are nonetheless collateral to the concerns of our thesis research, and will not be discussed.

In Subsection 2.1.1, the intuitive notion of a reference retrieval system is explicated at the top level, introducing the most important terms to be used in the remainder of the section. In the second subsection, we provide a brief overview of the general organization of past and present retrieval systems. Subsection 2.1.3 focuses special attention on the forms of document description used in typical existing reference retrieval systems. Subsection 2.1.4 is concerned with the kinds of pattern descriptions that may be formulated by the user in various retrieval systems, and with the ways they may be matched with the stored descriptions of documents in the collection.

In the next subsection, we will consider the problem of assessing the utility of a reference retrieval system. Since many different dimensions of performance may be significant in evaluating the assets and liabilities of a system, Subsection 2.1.5 will consider the kinds of *performance criteria* that may be relevant to the evaluation of reference retrieval systems. In Subsection 2.1.6, we will review a number of *evaluation methods* that have been used or suggested for assessing retrieval system performance.

Based on this foundation, we will consider in Subsection 2.1.7 the nature and significance of the newly emerging market for *online* reference retrieval systems and services, followed by a rather detailed outline of the sorts of features actually found in currently operational online retrieval systems, in Subsection 2.1.8. Subsection 2.1.9 discusses a few of the most apparent recent trends in reference retrieval system research. Finally, a sample interactive session with the DIALOG system, used as an example at various points throughout this section, is presented in Subsection 2.1.10.

2.1.1 Fundamental elements of the retrieval system

For purposes of our current discussion, we may describe the central functions of an information retrieval system as the *indexing* and *retrieval* of information *targets* within a (possibly dynamic) target *collection*. In many cases, the targets actually manipulated by the retrieval system do not themselves include the information sought by users, but instead bear a one-to-one correspondence with a set of *documents* containing the information of ultimate interest, and themselves contain information facilitating:

1. user assessment of the potential relevance of the document to his or her needs, and
2. location of the document itself.

Such *reference* targets might, for example, contain bibliographic information, a brief abstract, and a reference to the shelf or microfilm storage location in which the document could be found. Because of the one-to-one correspondence between the set of references and the documents to which they refer, we will use such terms as "target", "collection", etc., freely in this section to refer either to actual documents or to the corresponding references, except in those contexts in which this relaxation of terminology would introduce confusion.

In those systems of interest to our discussion, some form of *target description* is formulated for each of the documents stored as part of the collection during the process of indexing, and that description incorporated as part of the reference to that document. In the course of retrieval, on the other hand, some form of *pattern description* will be constructed by the user by way of expressing certain relevant characteristics of the desired targets. (These descriptions should be regarded as analytic constructs which may or may not have an obvious physical realization in any particular information retrieval system. In

many cases, the pattern or target descriptions may be implicit in the operation of the system, and not explicitly specified in the user request or in conjunction with the target record.)

Apart from a number of support functions, including the facilitation of both kinds of description-building and the provision of assistance to the inexperienced user, the central function of a reference retrieval system may be viewed as the identification of one or more target documents whose descriptions are assessed as consistent with the pattern description constructed by the user. Depending on the particular system, this function may be accomplished either by indicating a set of references deemed relevant to the user's request, or by ranking some or all of the documents according to their relative degree of assessed relevance.

In order to implement this function, most nontrivial traditional reference retrieval systems utilise some form of *index* in which the correspondence between potential search requests and candidate targets is manifested. In the course of search, a set of (human or mechanical) *retrieval procedures* typically refer to this index in order to locate an "appropriate" target set. Subject cards in library catalogs, sets of edge-punched cards, and inverted computer files are examples of the indices of three common types of retrieval systems.

Traditionally, a distinction has been drawn between the *indexing* and *classification* of documents. The term classification has been applied to the process of assigning a *single* (though possibly hierarchical) descriptive designation, such as a Dewey Decimal or Library of Congress number, to a given document, while *several* descriptive terms might be specified in the course of indexing, as in the specification of a number of distinct subject categories in a library card catalog. (In point of fact, even the card catalog has incorporated a classification-like indexing assymetry in its reliance upon a single "main entry" card, which tradi-

tionally contained more information than all other cards for the document in question. The present value of this convention, dating from the time when catalog cards were individually typeset, and not computer-generated, has been questioned by recent authors [Harris, 1974].) In the language of Section 2.2, classification is thus a process of description by simple or hierarchical abstraction, while indexing involves the use of multiple abstractive mechanisms.

Prior to the introduction of computer-based retrieval systems, this distinction was justified primarily by the need to assign a single description (a classification) to a document for purposes of determining its physical location within the library shelving scheme, while still permitting multiple access paths to the document based on different aspects of its content using different cards in a card catalog (an index). In a reference retrieval system, however, the physical storage location of a reference record does not have the significance of the location of a document in a library; the reference may be retrieved using any of several index terms, obviating the need for a distinguished classificatory term. In all fairness, it should be acknowledged that there are many purposes for which the use of a single "classifying" descriptive term is still valuable. Such a scheme is required, for example, if all documents are to be listed without redundancy in a hierarchically organized list (in certain thesauri, for example). In a discussion of contemporary computer-based reference retrieval systems, though, it is the notion of indexing, and not classification, that is most salient.

Indexing based on user-formulated descriptions of document content comprises only one of the techniques by which documents of interest to a user may be located, and is in fact made less attractive by the need for human effort in formulating the descriptions used in retrieval. It would thus seem appropriate to consider, at least briefly, a set of interesting techniques for accessing relevant documents on the basis of information associated with the

original, unindexed document. Kessler [1961, 1963] has proposed and tested a method for decomposing a large number of papers into groups on the basis of the publications they reference in their bibliographies. This technique, which he called "bibliographic coupling", is based on the assumption that the common citation of a given paper by two or more authors tends to indicate a relationship in content between the citing papers. Kessler assigns papers to a common group whenever they reference at least one paper in common with a given "test paper", and proposes a measure of "coupling strength" between a member of the group and the test paper for that group which is equal to the number of references shared by the paper in question and the test paper. Interestingly, groups of papers classified using the bibliographic coupling technique in fact evidence a strong intuitive relatedness in content. In a test involving 334 papers in Volume 112 of *Physical Review*, the technique of bibliographic coupling showed a high correlation with the classifications assigned by Analytic Subject Indexing by editors of the journal [Kessler, 1965].

The specification of a set of criteria for evaluating the performance of a particular reference retrieval system constitutes one of the most significant outstanding problems in the field, and will form the basis for most of the discussion in Subsection 2.1.5. To provide a context for the discussion which follows, however, it will be useful at this point to describe in general terms two constructs that play central roles in virtually all discussions of retrieval system evaluation: *recall* and *precision*.

These two criteria have been explicated in various ways by different authors, and our own concerns will in fact require a fundamental reassessment of their meaning and import in the context of knowledge-based retrieval. At this point, we will attempt only to convey a brief intuitive impression of the common use of these two terms. Generally speaking, recall is a measure of the extent to

which "appropriate" targets are retrieved, while precision reflects the extent to which "inappropriate" targets are excluded from the result of a search. Empirical data, along with certain theoretical arguments, have been generally felt to reveal an inherent tradeoff between the optimisation of recall and precision, at least within the general design framework of current retrieval systems.

From the perspective of knowledge-based retrieval research, our principal concerns are related to the characteristics of the descriptions and description-matching mechanisms used in existing reference retrieval systems. These issues define the central focus of Subsections 2.1.3 and 2.1.4. To provide a context for these discussions, though, we will first provide a brief sketch of the general organization of past and present reference retrieval systems.

2.1.2 Organization of retrieval systems

Since the 1950's, many taxonomies of reference retrieval systems have distinguished between two types of system organisations: *item entry* and *term entry*. (The use of these terms varies somewhat in the literature, and different terms have been used by different authors to describe essentially the same distinction, while others have adopted several additional categories. The basic notions of item vs term entry, though, provide a useful tool in describing traditional information retrieval schemes.)

In general terms, an item entry system may be described as an information retrieval system whose index is composed of a number of item records of some form, each corresponding to a single target document in the collection. Information describing a given document is associated with its item record at the time of indexing.

The index of a term entry system, on the other hand, is composed of *term records*, each of which corresponds to a single descriptive element, and includes information about the documents with which this descriptive element was associated. (We have used the phrase "descriptive element" in place of the term "descriptor", used more commonly in the literature of information retrieval to avoid confusion with a more specific use of the latter term in our own work, as introduced in Section 2.3.) The form and interpretation of these descriptive elements varies widely among the many examples of term entry systems, and will be discussed in Subsection 2.1.3.

These two types of file organisation may thus be distinguished by the structure of their hierarchical indices: in an item entry system, descriptive elements are hierarchically subordinated to the targets to which they are applicable, while in a term entry system, the order of subordination is reversed.

The distinction is illustrated graphically below.

Item entry organization:

document number	*associated descriptive elements*
1	*A, C*
2	*C*
3	*A, B, C*
4	*A, B*

Term entry organization:

descriptive element	*associated document number*
A	1, 3, 4
B	3, 4
C	1, 2, 3

Marginal hole (or edge-notched) card systems and their larger-scale functional equivalents exemplify the item entry organization method. The index in a simple version of such a system is comprised of a set of cards, one corresponding to each document in the collection, and each having a number of holes close to, but not touching, the perimeter. In the simplest (though often inefficient) encoding method, each edge position corresponds to a single descriptive term, and is punched out if applicable to the document in question, thus converting the hole to a slot extending to the card edge. When rods are passed through the holes corresponding to one or more descriptive terms, only those cards

corresponding to items matching all of the terms remain unsupported, resulting in the retrieval of targets corresponding to a simple conjunctive pattern description (see Subsection 2.1.4).

Simple examples of traditional term entry systems include such interior hole card systems as Peek-a-boo, Insite, Ekaba, Delta, etc. Although a variety of media and mechanisms are used in these systems, in each case the index is composed of a number of cards (or alternately, tapes, or some similar surface), each corresponding to some descriptive term potentially applicable to one or more documents in the collection. On the interior surface of each term card, holes are punched at the particular locations assigned to those documents to which the term in question is applicable. When the cards corresponding to several terms are superimposed, the set of documents to which each term is applicable may be observed. Another example of a term entry system is Uniterm (Taube, 1953), in which the index is composed of a set of specially ordered lists of document numbers, each list corresponding to a descriptive term. In the course of a search, the document lists corresponding to one or (in the case of *coordinate indexing*) more terms may be visually scanned by the user to find a potentially relevant set of documents.

While the item/term entry distinction was originally applied to the classification of mechanical and electro-mechanical retrieval systems, many of the early computer-based reference retrieval systems can be viewed either as straightforward applications of one or the other method to the organization of a hierarchical computer database or as derivative or hybrid variants of these methods.

The earliest computer-based information retrieval systems that found extensive practical application were fairly straightforward adaptations of the item entry method of file organization. Most of these early systems utilized se-

quential file processing techniques in which individual item records from the sequential index file were transferred from secondary to primary storage and tested for consistency with a given set of descriptive terms. The influence of item entry organization can also be seen in certain contemporary database management systems based on hierarchical data models.

Computer-based inverted file systems, on the other hand, exemplify the use of term entry organization. One simple design for an inverted file system utilizes an index comprised of one linked list for each descriptive term. The elements of each list are pointers to the documents to which the term in question is applicable. To retrieve the set of documents corresponding to a given pattern description, list-based set operations are performed on the document lists corresponding to the various descriptive terms involved in the pattern description. In another variation, each term is assigned a fixed-length bit array, with each document represented by a single bit position, allowing the use of rapid, machine-supported logical set operations in locating the potentially relevant documents.

2.1.3 Description of target documents

As noted in Subsection 2.1.1, the process of indexing involves the association of a single target *description* with each of the documents stored as part of the collection. In this subsection we will consider the forms of a number of different kinds of descriptions used in past and present retrieval systems to characterize the target documents. It is, of course, impossible to fully assess the significance of various target document description schemes independently of the pattern descriptions and matching procedures incorporated in the system. Much of the import of this subsection will thus become clearer in Subsection 2.1.4.

Among the methods which have been used to assign descriptions to target documents are a wide range of techniques extending from highly subjective evaluations of content to tightly defined, replicable procedures. At the former extreme, the description might be assigned by a human indexer, possibly having expertise in either the subject domain or in the classificatory methods adopted for use in the system. The other end of the continuum might be represented by fully automated description construction based on a straightforward, computationally effective classifying procedure carried out by a computer.

The particular subset of the document text on which indexing is based also varies among the various existing document retrieval systems. In some systems, the whole text serves as a basis for (manual or mechanical) indexing, while in others, only the title, abstract, some initial or final portion of the text, or some other manually or mechanically extracted subset of the document, are examined for purposes of classification.

Perhaps the simplest classification scheme involves the selection of a fixed set of descriptive tokens prior to indexing. (We will use the term "token" to refer to a non-decomposable description -- one having no components whose in-

terpretation is preserved when used as a component of a different description. The motivation for differentiating such "atomic" descriptions from "compound" descriptions built from a number of indendently meaningful components will become apparent later in our discussion.) Tokens from this pre-specified set would then be used, either singly or in combination, to characterize the salient qualities of each document added to the collection. To be useful, the fixed set of tokens is ordinarily chosen carefully by an individual familiar with the behavior of the retrieval system and with the domain of discourse encompassed by the documents anticipated as members of the collection. These tokens may have the character of generic subject categories, significant keywords actually found in the texts to be catalogued, or other indicators facilitating the discrimination of documents in a manner useful to the user.

In practice, however, the pre-selection of a fixed set of descriptive tokens is found to introduce serious problems whenever the content of the document collection varies dynamically. In a typical fixed-token system, growth in the collection generally results in:

1 a proliferation of documents classified within a small subset of the fixed set of categories, and

2 a preponderance of problematic categorization decisions facing the indexer.

The problems associated with use of a fixed set of descriptive tokens are aggravated by such factors as ongoing changes in the subject domain or variations over time in the kinds of documents added to the collection.

One common alternative to the use of a fixed set of independently chosen descriptive tokens is the construction of target descriptions from one or more words actually found in the text or text subset, possibly modified according to certain pre-processing rules. Again, it should be noted that the text segment

from which such descriptions are derived may encompass either whole documents, abstracts, initial or final text portions, or other text subsets selected manually or mechanically according to a wide range of possible criteria. In the discussion which follows, the term "text" will generally be applicable to any such text or text segment.

A number of strategies have been employed in choosing the set of text words (or word derivatives) used for text-based indexing. One method involves the specification of a fixed (though often quite extensive) list of *keywords*. Each document in the collection is then assigned a description consisting of all of the pre-specified keywords that appear in the text. This form of indexing has been employed in both manual and computer-based systems, but is especially suitable for automation when the full text or a lengthy text segment is used for indexing. Moreover, the use of a list of keywords whose number and identity is fixed (if not perpetually, at least between discrete points of "system generation", which would typically occur infrequently), often allows the use of certain computer-based indexing techniques which offer significant savings in computation time and memory space. Since the keyword list used to index a given document is determined prior to its indexing, this method is still vulnerable to changes in the content of the document collection.

Still another mechanism for indexing involves the inclusion of *all* significant text words as part of the description. For reasons of both efficiency and ease of document discrimination, certain words which are not deemed useful in distinguishing documents in a meaningful way are generally excluded from use in such descriptions. Explicit exclusion lists typically include words that appear with extremely high frequency in most documents—prepositions and adjective articles, for example—and a number of other terms presumed to have little value in distinguishing documents consistent with user requirements. Excluded

word judgements have most commonly been made subjectively, although considerable attention has been devoted to the question of the utility of various sorts of text words in document discrimination. The KWIC (Key Word In Context) system [Luhn, 1959] may be regarded as an application of the principle of exhaustive text word indexing (where the text is in this case typically a title) subject to an explicit exclusion list. (Note that we have discussed only the *indexing* mechanisms of the KWIC system; the utility of KWIC is based in large part on its contextual facilities, which will not be discussed here.)

One of the modifications in descriptive form that has been employed experimentally in an attempt to improve the recall/precision characteristics of text word indexing involves the *differential weighting* of certain descriptive tokens to reflect their relative values in characterizing the content of a document. The use to which these assigned weights may be put in matching user requests with target documents is discussed in Subsection 2.1.4.

The ranking of text word descriptors in order of their frequency of occurrence in the text is one method by which such descriptive tokens have been weighted. Another method involves the computation of the frequency of occurrence of a given word in the text divided by the frequency of occurrence of that word in general usage, or in a large corpus of texts. The rationale behind this and other word frequency ranking techniques is that frequently occurring non-excluded terms are likely to be strong indicators of the content of a document, particularly in the case of terms that are rarely used in the full collection.

In some retrieval systems, descriptions have been built using multiple-word tokens. Pairs of adjacent words, for example, or contiguous phrases, have been used in building document descriptions in several experimental systems. The phrase "document descriptions" appearing in the previous sentence, for

example, might be treated as a single, non-decomposable token as part of the description of this document. In another variation, sequences of text words occurring in a particular *order*, even if non-contiguously, within a single sentence (or in some other portion of text delimited by pre-specified separators) may be treated as a token for purposes of indexing. The combination "sequences ... words", for example, might be incorporated in a description of this document on the basis of the content of the previous sentence.

It should be recalled at this point that multiple-word tokens, which are *indivisible* elements of document descriptors, have been defined in a way that distinguishes them from descriptions involving multiple tokens, each of which may be independently interpreted by the procedures involved in retrieval. To clarify this distinction, it may be useful to regard multiple-word tokens as "pseudo-text words" produced by the application of a straightforward lexical preprocessing routine, and subsequently treated in exactly the same manner as ordinary keywords extracted from the text.

While the use of actual text words has several obvious advantages as a method for description construction, including objective replicability and ease of straightforward automation, a high price is typically paid in the form of a low level of recall of "appropriate" targets. In a strict text word indexing system, for example, an intended target will be missed when the pattern and target words differ only in a syntactic suffix. (A pattern description incorporating the word *computers*, for example, will fail to retrieve a document in which the word appears only in the singular form *computer*.) In a number of text word indexing systems, the number of distinct descriptive terms has thus been reduced by mapping several syntactic forms of a word to a single canonical form. Although this convention has been employed with considerable success in both experimental and practical systems, the standardization of descriptive

terms even by eliminating syntactic variants has proven to be a non-trivial problem

Other methods of *vocabulary control*, as such techniques are often called, raise more troublesome questions One example is the problem of recognizing situations in which *different* words should be treated as identical in the interest of recall maximization In certain systems, this problem has been attacked by partitioning text words into disjoint sets which are treated similarly for purposes of classification and retrieval In some cases, only words judged to be "true synonyms" (the precise explication of which must be regarded as problematic, at best) have been treated identically in the course of indexing In other systems, a variety of conventions have been adopted for grouping together words that can be usefully mapped to a single term—although generally at the expense of a decrease in precision

One such convention, for example, involves the grouping of a number of words under a single more *abstract* term of which each word may be considered an instance (References in the literature commonly speak of the use of *generic* terms in discussing this technique; our terminology has been adopted in the interest of consistency with the notions of structure and abstraction, which will be introduced in Section 2 2) As an example of this technique of vocabulary control, use of the words *compiler*, *interpreter* and *assembler* in various computer science texts might all result in the choice of *translator* as a more abstract descriptive term One consequence of such a convention, of course, would be a loss of the capability to distinguish between these different types of language translators in the course of retrieval, potentially resulting in the selection of a large number of documents irrelevant to the user's requirement The use of more abstract descriptive tokens thus illustrates the tradeoff between recall and precision to which we have already alluded

A large proportion of the research efforts in the field of document retrieval has involved the search for document description schemes that more precisely characterize the meaning of each indexed document while minimising adverse effects on recall Although the common themes underlying these various mechanisms for attacking the recall/precision tradeoff problem will be more easily appreciated after some of the discussion which follows, it would be appropriate to remark at this point that most of these schemes are based on the use of multiple-token descriptions in which the different functions served by the constituent tokens are distinguished

One of the most primitive of these schemes involves the partitioning of the set of descriptive tokens into a number of *interlocking* sets [Mooers, 1956]. In such a system, user requests can specify that certain search criteria be satisfied by tokens all of which appear within a single interlocking set, but not by tokens from more than one set To provide a simple example of the principle of interlocking based on straightforward text word indexing, we will describe a hypothetical system in which all key word tokens derived from a given sentence are grouped together in a single interlocking set Consider, for example, the following abstract:

> *"A brief history of document retrieval is presented This survey is followed by a detailed examination of a contemporary system for the retrieval of documents from a large medical school library."*

This article might be described by the following two interlocking sets of tokens:

1. *(first sentence)*:

(*history, document, retrieval*)

2 *(second sentence)*

(system, retrieval, document, medical, school, library)

Such a partitioning would allow the use of a matching procedure which would retrieve the article in question in response to an inquiry for information about the history of document retrieval, or about medical documents, without retrieving all articles dealing with the medical histories of patients. In a system supporting the appropriate pattern descriptions and matching procedures (discussed in the following subsection), in fact, targets might qualify for retrieval on the basis of a conjunction of several tests, each satisfied by a different interlocking set. A request for articles discussing *both* the history of document retrieval *and* medical school libraries, for example, could be used to locate the above document, again without retrieving irrelevant articles discussing medical histories.

The technique of interlocking has in fact been employed in various forms in a large number of practical and experimental information retrieval systems. In a system developed at Western Reserve University for research purposes, documents were described by "telegraphic abstracts", each structured in a three level hierarchy of interlocking classes. (Members of the "lowest level" interlocking set, however, were not indivisible tokens, as in our simple example, but structured entities built from independently meaningful components.) An alternative realization of interlocking sets is embodied in the method of *interfixing* developed by the U.S. Patent Office [Andrews, 1959]. This method involves the association of a distinctive tag, called an *interfix*, with each element of a given interlocked set. In such a system, no explicit

list is maintained of the members of each interlocked descriptive set. Instead, the members of each such set are identified by tagging each one with a common symbol, called an *interfix*. (As before, we have ignored many interesting characteristics of the techniques developed by the Patent Office, focusing our attention solely on the mechanism of interlocking descriptive sets.)

One of the most important weaknesses of the sorts of descriptive mechanisms that we have discussed thus far involves the inability to distinguish the different conceptual roles served by various components of the document descriptions. An example offered by Bar-Hillel [1957] illustrates a very simple form of this problem: How can a document describing "the export of wine from Germany to France" be distinguished from one dealing with "the export of wine from France to Germany"?

The problem of distinguishing the various *roles* that may be served by the same token (or compound descriptive element) to avoid retrieving inappropriate documents in which a token appears in the wrong functional context has been approached in a restricted way in many information retrieval systems (see, for example, Costello and Wall [1959]). A theoretical framework for the notion of roles will be provided in Section 2.2. In this subsection, we will consider several examples of the use of simple role mechanisms in existing information retrieval systems.

Perhaps the simplest manifestation of role distinctions in document descriptions involves the specification of descriptive components in a *linear sequence* of predetermined order. The order of appearance of each such descriptive element may then be used to determine its role in the description. The two documents in Bar-Hillel's example, might be distinguished by the order in which the key word tokens *Germany* and *France* appeared in the description. The distinction between points of origination and destination is of course applicable

only to a small subset of all possible document descriptions. By specifying a small set of linear sequencing rules that distinguish roles in a number of common situations, though, a certain degree of document discrimination can be achieved by comparison with the use of unordered sets of descriptive elements.

More powerful role-distinguishing mechanisms are possible if symbols are included in the description to explicitly indicate the roles of and/or the relations between constituent descriptive elements. One form of explicit role distinction involves the inclusion of "preposition like" connective elements in the description. Bar Hillel's first document might be encoded as

$$(export, OF, wine, FROM, germany, TO, france) \quad ,$$

for example. An alternate method of explicit role distinction involves the "inflection" of descriptive elements by attaching "case indicators" to each such element according to its role, as in certain natural languages. This mechanism (which in fact forms the basis of the role distinction capabilities of the Western Reserve University system discussed earlier) might be used to encode the example document as

$$(export\ ACT, wine\ OBJ, germany\ ORIG, france\ DEST) \quad .$$

Typical systems employing the principle of explicit role denotation have supported between five and thirty different "canonical" role indicators reflecting a small number of relationships felt to be useful in describing many members of the document collection at hand

2.1.4 Retrieval of matching documents

In this subsection, we will review the forms of pattern descriptions that may be specified by the user in various reference retrieval systems, along with the matching criteria used to assess the applicability of candidate targets to such pattern descriptions. We will be primarily concerned with the functional behavior of alternative matching schemes, and not with the detailed procedures by which such mechanisms are realized. A unified theoretical analysis of these mechanisms will be deferred to Section 2.2.

The simplest approach to pattern description and retrieval is based on the user specification of a single descriptive token which must exactly match the description associated with any retrieved target. While this mechanism is not powerful enough for actual use in most reference retrieval applications, the single-token pattern description will serve as the basis for a simple illustration of certain variants of the exact match criterion that will be relevant to practical multiple-token pattern description systems as well.

The first such extension, which we will call the *simple abstractive match*, provides for the retrieval of all targets whose associated description either exactly matches or may be considered a special case of the single token pattern description. A computer science article bearing the descriptive token *pascal*, for example, might be retrieved not only by the identical pattern description, but by the generic descriptive token *language* as well. Another extension allows the retrieval of more abstract targets as well as those bearing a more specialized descriptive token. Under certain circumstances (the lack of any target bearing the token *pascal*, for example), the pattern description *pascal* might thus retrieve all references associated with the more abstract keyword *language*. Other variants provide for the retrieval of targets described by

tokens designated (either by a human cataloguer or through the application of mechanized procedures such as statistical clustering and concept association techniques) to be *related* to the pattern description

To this point we have discussed only single-token pattern descriptions—descriptions composed of a single indivisible meaning-bearing descriptive element. Most practical retrieval systems, though, permit the use of *compound descriptions* built in various ways from one *or more* of these atomic descriptive elements. The first type of compound pattern description that we will consider is the *boolean pattern description*, constructed by applying logical connectives to primitive tokens. In particular, the boolean combinations of tokens used in most document retrieval and generalized database management systems are built from the following operators.

1. The logical conjunction operator $\wedge$
2. The logical disjunction operator $\vee$
3. The logical negation operator $\sim$

The simplest sort of system allowing the user specification of boolean compound descriptions is based on the association of multiple descriptive tokens with each target document to reflect *all* subjects discussed in the document. For the sake of conceptual symmetry, it may be useful to think of both the pattern and target descriptions as a boolean combination of descriptive tokens, but with the target descriptions restricted to the use of only the conjunction operator. Target descriptions involving more than two tokens may be derived by applying the conjunctive operator recursively to existing compound descriptions as well as primitive tokens. A target description including the tokens A, B and C, for example, may be interpreted as a two-level conjunction of the

form

$$(A \wedge (B \wedge C))$$

The interpretation of boolean pattern descriptions by the matching routines in our simple system is defined by the following rules: A compound pattern description built by conjoining two tokens A and B with the *AND* operator is interpreted as matching any target whose description includes both A and B. The disjunction $(A \vee B)$, on the other hand, matches any target description bearing either token A or B or both. Finally, the description $\sim A$ matches any target whose description *does not* include the token A.

Furthermore, multi-level boolean pattern descriptions may be constructed by applying the three logical operators to existing compound descriptions as well as primitive tokens. Generalising the above definitions accordingly, we can specify the following recursive criteria for matching a boolean pattern description against a multiple-token target description:

1. A target token is consistent with a pattern description if the token is included in the pattern description.
2. A conjunctive target description $A \wedge B$ is consistent with a pattern description if the subdescriptions A and B are both consistent with the target description.
3. A disjunctive target description $A \vee B$ is consistent with a pattern description if either of the subdescriptions A or B, or both, are consistent with the target description.
4. A negated target description $\sim A$ is consistent with a pattern description if the subdescription A is inconsistent with the target description.

Typical of the use of boolean compound descriptions is the target description

$$(cache \vee (pipeline \wedge (\sim oil)))$$

which might be used in a computer science reference retrieval system in an attempt to retrieve references to all documents related to the use of cache memory or pipelining techniques in fast CPU design without retrieving articles related to the use of computers in, say, planning the routes of oil pipelines.

Although the set of three boolean connectives introduced above is sufficient for the construction of any logical combination of operands (in fact, there exist *single* logical operators that themselves form such a basis, but whose interpretations are less useful in the context of document retrieval), certain additional logical operators are sometimes included for convenience. The *exclusive or* operator (sometimes denoted *XOR*), which when applied to tokens *A* and *B* yields a compound description matching any target bearing either *A* or *B* but *not both*, is an example of such an additional boolean connective.

Another compound pattern descriptive mechanism supported by many current retrieval systems involves the explicit specification of roles filled by constituent elements of the description through the use of *attribute/value* pairings. The attribute/value mechanisms used in existing document retrieval systems in fact represent a highly restricted application of a very powerful construct which will be discussed at some length in Section 2.2. In this subsection, we will attempt only to convey a feeling for the most common use of such mechanisms in current systems.

As it is most commonly used in document retrieval, this mechanism involves the specification of a small, fixed (from the viewpoint of the user) set of *attributes* common to most documents in the collection. Typical attributes might include an *author*, *title*, and *publication date*, along with a single *subject* attribute. In describing a document, appropriate *values* are associated with each attribute, as described in the discussion of roles in Subsection 2.1.3. In the course of search, the user formulates separate subdescriptions corresponding to one or more attributes that must be satisfied by the values of the same attributes in a matching target description. Attributes for which no value is specified in the pattern description are considered consistent with any value that may be associated with the attribute in the target description. The subdescriptions associated with individual attributes may be either primitive descriptive tokens or, in many systems, compound boolean descriptions. (The recursive appearance of attribute/value subdescriptions as values of "higher-level" attributes, while a potentially significant tool, has to our knowledge not yet been incorporated in an operational reference retrieval system.)

In a retrieval system utilizing attribute/value and boolean combinational mechanisms, for example, the pattern description

author = *winograd*
subject = (*artificial intelligence* ∧ *knowledge representation*)

would retrieve the target description

author = *winograd*
title = *five lectures on artificial intelligence*
subject = (*artificial intelligence, natural language processing, knowledge representation, automatic programming*)
date = 1975

In addition to exact match tests, special attribute-specific tests are often provided for testing the particular types of values associated with different attributes. Numerical comparisons, for example, can often be specified as part of the pattern description to test values for consistency with a *range specification* in place of a specific numerical value. The attribute/value pair

date ≥ 1972

for example, might be specified in place of the explicit year specification

incorporated in the above example, and would be considered consistent with either 1972 or any subsequent year associated with the *date* attribute of a target document. Special routines for processing character strings are among the other sorts of special attribute-specific procedures that may be incorporated in the matching schemes of an attribute/value retrieval system.

Many variants of the pattern description and matching methods described above are based on the numerical *weighting* of individual tokens in either the pattern or target descriptions, or both, and the evaluation of some mathematical function sensitive to these variable weights in the course of retrieval to rank candidate targets in an attempt to estimate the *goodness-of-match* of a potential pattern/target pairing. In a typical target-weighting scheme, individual tokens are assigned weights intended to approximate their relevance to the target document (as estimated, for example, by their relative frequency of occurence in the text or text segment upon which indexing is based), their expected usefulness in subsequent matching, or some other property useful in the particular system. The specification of a pattern term matching a highly-weighted target term would then make a more significant arithmetic contribution to the goodness-of-match value computed for the pattern/target pair under consideration than the specification of a term matching a lower-weighted pattern term. In a similar manner, some systems allow the user to assign weights to individual components of the pattern description to reflect the relative importance of these constituents. The simplest weighted matching schemes involve the assignment of weights to *either* pattern *or* target tokens and the computation of goodness-of-match as a linear combination of term weights. Targets are then ranked in order of the value of this function, and an ordered list of candidates returned to the user. A variety of more complicated weighting techniques and matching algorithms have been explored experimentally.

2.1.5 Criteria for retrieval system evaluation

Because of the great diversity of purpose and design found among reference retrieval systems, the specification of a simple and uniformly applicable set of system performance evaluation criteria is a difficult task. The utility of a given evaluative tool is highly dependent on such system parameters as the volume of retrieval activity, the size of the document collection, the time scale on which retrieval takes place, and the number of users. In spite of such difficulties, a number of widely applicable evaluation criteria have been proposed and critically examined by various researchers, especially during the past decade.

A number of taxonomies of the general categories of criteria relevant to retrieval system evaluation have been proposed by various authors. One such breakdown, based on a discussion by Lancaster and Fayen [1973], follows:

1. *Recall*. The proportion of relevant articles in the collection that are retrieved by the system.
2. *Precision*. The proportion of the retrieved articles that are in fact relevant.
3. *Coverage*. The extent to which the collection includes documents relevant to a particular area of inquiry.
4. *Response time*. The time typically required by the system to respond to user requests.
5. *User effort*. The ease with which the system can be used.
6. *Output form*. The characteristics of the final results and intermediate feedback provided to the user.

The discussion below is primarily concerned with the characteristics of the first two criteria for system performance evaluation.

The kinds of operational measures that may be adopted to assess the degree of recall and precision are dependent on the type of information provided

in the search output. This dependency can be best explained by viewing the result of a search process as the imposition of an *ordering* on all documents in the collection. The most informative ordering that may be imposed on the collection is a *total ordering*, which insures that for any pair of documents in the collection, the one with greater assessed relevance may always be determined. A total ordering may be thought of as a one-to-one correspondence between the n documents in the collection and the integers 1 through n.

In most systems, however, only a *partial ordering* is imposed on the collection, such that the more relevant document may be determined for only some, but in general not all, of the possible document pairs. In such systems, less information is available for the computation of performance measures. The most common example of partial ordering results only in a dichotomous partition of the collection into relevant and irrelevant documents, so that the "more relevant" relationship is defined only when one document is a member of the former set and the other belongs to the latter. No assessment of the comparative relevance of two "relevant" or two "irrelevant" documents is available for computing performance measures in such systems.

A somewhat more informative partial ordering is imposed on the collection in systems which *rank* all documents deemed relevant in order of their *degree* of relevance, thus allowing relative relevance comparisons between pairs of relevant documents or between one relevant and one irrelevant document, but not between pairs of irrelevant documents. By the (user-specified or implicit) specification of a numerical *cutoff* point, some or all of the relevant documents may be presented to the user, often in order of their degree of relevance. Apart from the facilitation of variable cutoff points, though, the ranking of relevant documents provides information that is used in various measures of system performance.

Simple ranking of relevant documents is by no means the only form of partial ordering that may be used. Evaluative procedures applicable to general partially ordered outputs (including total ordering as a special case) have been proposed [Hocchio, 1964].

Even more information than that provided by a total ordering may be available for purposes of performance evaluation when it is possible to define a *metric* on the set of documents in the collection. In a simple one-dimensional scheme, for example, each document might be assigned a real number reflecting its degree of relevance to the user request at hand. In summary, performance measures may themselves be ranked according to the output on which they are based, from most to least informative:

1. Measures based on a relevance metric
2. Measures based on a total ordering of assessed relevance
3. Measures based on a partial relevance ordering more constrained than a simple dichotomous partition and less constrained than a total ordering
4. Measures based on a simple dichotomous partition between relevant and irrelevant documents

For simplicity, we will first review the sorts of recall and precision measures that have been used or proposed for evaluating systems that impose only a dichotomous partition on the document collection. Most of these measures can be derived from the quantities a, b, c and d, defined as follows:

a	*(hit)*	References retrieved and relevant
b	*(waste)*	References retrieved, but not relevant
c	*(miss)*	References relevant, but not retrieved
d	*(dodge)*	References neither retrieved nor relevant

The earliest measures derived from these quantities were proposed by

Perry, Kent and Berry [1956]. These authors defined the following measures of system performance:

Resolution: $(a + b)/n$

Elimination: $(n - a - b)/n$

Pertinency: $a/(a + b)$

Noise: $b/(a + b)$

where n is the number of references in the collection. Resolution and elimination are measures of the proportion of the collection retrieved in the course of search, without regard to the relevance of the retrieved documents. Pertinency and noise, on the other hand, reflect the proportion of retrieved references which are relevant and irrelevant, respectively, and thus capture the essence of the notion of precision introduced in Subsection 2.1.1.

The most widely used measures derived from the quantities a, b, c and d, however, were introduced by Cleverdon [1962] in the context of the Cranfield Project, which will be discussed in Subsections 2.1.6 and 2.1.7:

Recall ratio: $100a/(a + c)$

Relevance ratio: $100a/(a + b)$

Cleverdon's recall ratio and relevance ratio correspond closely with our intuitive notions of recall and precision, respectively, and have been used as the point of departure for a great many further investigations involving both partition- and ranking-based performance evaluation measures.

It should be noted that the values of the classical partition-based recall and precision measures described above are highly dependent on the number of references retrieved. If all potentially relevant references are presented to the user, recall will in general be higher, and precision lower, than would be the case if only the most relevant documents were retrieved. In the language of ranked-output systems, Cleverdon's recall and relevance ratios are highly dependent on

the cutoff point chosen for measurement. Characterizations of the behavior of a retrieval system in terms of such cutoff dependent partition-based measures generally have the form of a set of recall/precision pairs, or a continuous curve derived from the locus of actual recall/precision values. Methods for constructing useful performance curves are outlined by Keen [1971]. Other suggestions for plotting one performance-related variable against another to capture certain essential characteristics of the performance of a given retrieval system, have been advanced by Mongomery and Swanson [1962], Swets[1963], and others.

Some researchers (Swets [1969], for example), have argued that an effective performance measure should be independent of any particular cutoff point, and should return a single numerical value when applied to a given retrieval system. One class of single number measures is exemplified by the *rank recall* and *log precision* quantities, and the related *normalized recall* and *normalized precision* measures [Salton, 1965; Salton, 1966, Rocchio, 1965; Rocchio, 1966]. These measures, applicable to retrieval systems that impose a total ordering on the collection, are based on a comparison of the actual ranking assigned by the system in response to a given search request with an ideal ranking based on the relative relevance that would be assigned by the user if each document in the collection were examined. While systematically related to the number of relevant references, such measures have been considered cutoff independent "in that the rank positions of all the relevant documents to a request are compared with the ideal positions resulting from a perfect system" [Keen, 1967]. Other single-number measures are discussed by Swets [1963] and Giuliano and Jones [1966].

Rocchio [1965] has emphasized the importance of the choice of averaging method used in computing a system performance measure on the basis of

multiple-query data. He distinguishes between *macro evaluation*, in which the measure in question is averaged on a per-query basis, and *micro evaluation*, based on per-document averaging. Instances in which significantly discrepant results are obtained from these two types of measures are presented, and the appropriateness of each for various purposes is discussed.

2.1.6 Evaluation methods and results

A number of strategies have been employed to evaluate the merits of existing retrieval systems and techniques. It is the intent of this subsection to briefly outline these techniques, and to provide illustrative examples of their application.

Among the simplest evaluation results that have been published are *comparative cost analyses* of two or more retrieval systems. Although the determination of useful cost data is complicated by the subjectivity of the choice of such variables as the amortization periods of different pieces of equipment, a number of interesting cost studies have been published and subjected to critical review by the retrieval system community. In one of the earliest such studies [Lockheed, 1959], researchers at Lockheed Aircraft Corporation compared the unit cost of executing a document search using a manual term-entry system, a punch card scanning system, a magnetic tape scanning system and a disk-based system.

Because of differing ratios of amortized fixed costs to incremental search costs for the four systems, different systems proved most economical depending on the size of the collection and on the volume of search activity. In particular, the rank ordering of the four tested systems was found to change several times as collection size and transaction volume was varied from 10,000 documents and 10 searches per day to 100,000 documents and 100 searches per day. Other comparative cost analyses have revealed similar patterns, in which different systems appear most economical in different ranges of operational scale as a result of tradeoffs between fixed and marginal costs.

Among the most useful evaluative tools is the comparative survey, typically involving a "checklist" analysis of the corresponding features in a number

of existing systems. Perhaps the most comprehensive such study was the survey of eighty-seven nonconventional information retrieval systems conducted by the National Science Foundation [1962]. Among the parameters specified for each of these systems were the subject area of the collection, the type of document indexed, the categories of users and operators involved, the document descriptive techniques employed, the kind of equipment used in the system, the processing operations supported by the system, and the size and cost of the system.

The NSF survey provided a general impression of the sorts of applications for which various kinds of retrieval systems seemed best suited. Additionally, the abundant corpus of quantitative data presented in the study later served as the basis for investigations of the relationships between certain critical system parameters within a given system.

Fife, et al. [1974] has compared forty-six interactive retrieval systems, some of which were oriented toward the retrieval of documents. Minker [1977] provides a tabular comparison of five typical document retrieval systems (MEDLARS II, RECON/STIMS, ORBIT II, the Defense Documentation Center Information System, and the New York Times Information Bank), adapted from material provided by Fong [1971] and Brandhorst and Eckert [1972]. Minker's data is organized according to the analytic framework that he has introduced to distinguish the features of reference retrieval systems, generalized database management systems and question answering systems.

Another method of performance evaluation involves the comparative testing of different retrieval systems or methods using a well-defined experimental retrieval task. The first phase of the Cranfield Project [Cleverdon, 1960] exemplifies the use of a comparative experimental procedure in evaluating retrieval system performance. In the course of this project, four reference

retrieval systems using different descriptive languages and file organizations were each used to index the same collection of 18,000 documents. A fixed set of 1200 search requests describing documents known to be in the collection were presented to each of the four systems. A number of independent variables were manipulated—some relevant to the primary interest of the experimenters, and others to assess the potential effects of variations extraneous to those interests—and the results were analysed in various ways, both by the Cranfield investigators and by a number of subsequent authors. In a subsequent phase of the Cranfield Project, a retrieval system developed at Western Reserve University was evaluated in a similar manner.

One important issue that must be faced in engineering such a comparative test involves the origin of the search queries posed to the systems being tested. In the Cranfield studies, technical personnel generated *artificial* search requests describing particular documents in the collection that were then reviewed by experienced librarians to eliminate those that seemed implausible. Such procedures simplify the assessment of search results, since at least one document (the one from which the query was derived) may always be considered relevant to each search request. It is not clear, however, to what extent artificial search requests should be considered a useful approximation of actual system use. This concern can be eliminated by the use of *natural* search requests generated by actual users having no prior exposure to individual documents in the collection at the expense of complications in assessing the appropriateness of search results.

Recall ratios in the Cranfield studies were found to range between 60 and 90 for all four of the systems that were tested, and under all experimental conditions, with the *mean* recall ratios for each system (averaged over all other independent variables) lying in the surprisingly narrow range of 74 to

82. A detailed failure analysis revealed that most instances in which a relevant reference was not retrieved were attributable to errors on the part of the human indexer. Among the other causes of recall failure, inherent inadequacies in the indexing and retrieval systems themselves were found to account for only 6% of the missed documents. The preponderance of user-attributable errors by comparison with internal system failures illustrates the importance of careful human engineering of information retrieval systems to facilitate the process of correctly indexing and searching.

The second major evaluative effort of the Cranfield Project team, in which the performance of the Western Reserve University system was compared to that of a faceted index designed by the Cranfield group, yielded results fundamentally consistent with their earlier four-system comparison. Recall ratios were found to be remarkably close to those in the earlier study—between 75 and 82, averaged over a number of experimental conditions. In the W.R.U. study, however, additional information related to the issue of precision was obtained by rating the relevance of each document to each question (on a scale of 1 to 4) prior to the tests and computing relevance ratios (see Subsection 2.1.5) for each relevance cutoff point. Relevance ratios in the test were all considerably lower than the corresponding recall ratios—under 30 in all cases, and generally under 20.

Of particular interest are the results of an analysis of the causes of erroneous retrieval of irrelevant references in the Cranfield W.R.U. study. Errors on the part of human indexers and searchers accounted for a much smaller proportion of the set of precision failures than was the case for recall failures. Inherent faults in the indexing language, on the other hand, were deemed responsible for the retrieval of 30% of the irrelevant references. Many of the other failures appeared to be based on such problems as over-exhaustive in-

dexing or overly general pattern descriptions, whose correction—within the constraints imposed by the system design—would seem to be directly and inextricably associated with a decrease in recall.

A number of experiments [Rath, Resnick and Savage, 1961; Swanson, 1961; Montgomery and Swanson, 1962] have attempted to compare the effects of various forms of target document description (see Subsection 2.1.1) on recall and precision measures. Among the descriptive forms that have been studied in these tests are simple subject headings, document titles, various forms of abstract (including actual abstracts composed by human authors, sets of statistically selected text sentences, and initial and final text segments), and the full text of the documents themselves.

Measurable, user-relevant system characteristics such as recall and precision behavior are not the only aspects of retrieval system performance of interest to researchers. On the contrary, much of the effort that has been expended in evaluation has focused on the behavior of *internal* system parameters relevant to the effectiveness and efficiency of the retrieval process. Considerable attention, for example, has been given to the relative frequency of occurrence of individual descriptive tokens within both target and pattern descriptions. Such *vocabulary use* statistics have been collected for a number of operational systems [Schultz, 1959; Melton, 1962; Houston and Wall, 1964] and several mathematical models have been proposed to explain observed relationships among the data. The evaluation of vocabulary use has provided information useful in designing optimal description encoding and search strategies. Other examples of internal system parameters that have been the subject of empirical investigation include CPU utilization, primary memory use, and secondary storage access statistics, in computer-based systems.

Several powerful subjective tools have been used to develop a stronger

qualitative understanding of the essential aspects of retrieval system behavior. One such tool, used in both phases of the Cranfield Project, is the set of procedures that has been termed *failure analysis*. 495 misses (instances of relevant documents not retrieved) were analyzed in the first phase of Cranfield to assess the reasons for recall failure. Failures were individually characterized using a simple two-level hierarchy describing the most common pathological conditions found to interfere with recall in the four systems tested. Cranfield researchers also applied recall failure analysis to the results of the W.R.U. system tests to investigate 111 cases in which an artificial search request failed to retrieve either the document from which it was derived or one judged relevant to the query. (The results of the Cranfield failure analyses will be discussed in Subsection 2.1.7.)

More recently, failure analysis has been used in evaluating the performance of online retrieval systems. In the evaluation of the MEDLARS system, as applied to the AIM/TWX (*Abridged Index Medicus*) database [Lancaster, 1972], for example, the causes of both recall and precision failures were investigated. The SPIRES system [Parker, 1970], an online interactive generalized database management system that has been successfully applied to various document retrieval tasks, has the capability of monitoring errors and other anomalies occurring in the course of a user session, providing information valuable in identifying and correcting the sources of common failures.

Another subjective method for system performance evaluation involves the elicitation of user feedback. Such feedback may be oriented toward satisfaction measurement, the isolation of problem areas, or other aspects of system evaluation. Techniques for collecting subjective user evaluations include interviews, the administration of questionnaires, and detailed case studies. Different methods vary widely in the degree of constraint imposed on the form of feed-

back elicited from the user, and in the sorts of measurement tools available for analyzing this feedback. In the evaluation of RECON by Bunker-Ramo [Meister and Sullivan, 1967], for example, an *evaluative rating form* is employed which solicits the user's subjective assessments of various aspects of system behavior, as expressed on a seven point bipolar semantic differential scale.

Freer forms of user feedback were collected in two evaluations of Lockheed's DIALOG system. In one study conducted for NASA [Summit, 1968], user comments and reactions were obtained to supplement a collection of operational statistics. In a second study [Timbie and Coombs, 1969; Summit, 1970] at the Educational Resources Clearing House (ERIC) Clearinghouse on Educational Media and Technology, a small group consisting of nine (as compared with thirty-seven, in the RECON evaluation) professional users were each interviewed in an extended *debriefing session* to record their experiences using the system to satisfy genuine information needs. These sessions were guided by a long list of questions posed by the interviewer, and were tape recorded and transcribed for analysis. The collection of user feedback using questionnaire, interview and case study methodologies has played a part in the evaluation of many other systems, including AUDACIOUS [Freeman and Atherton, 1968] (tape recorded user reactions), SUPARTS [Cook, et al., 1971] (user interviews), DDC [Bennerts, 1971] (guided user interviews), and EARTS [Lancaster, 1972] (multiple choice and short textual response questionnaire).

One successfully-employed evaluative mechanism applicable to many computer-based retrieval systems is the online collection, and in some cases, analysis, of data relevant to performance evaluation. In some applications, information is openly solicited from the user, in other cases, aspects of system performance are measured *unobtrusively* in the course of a user session.

One of the most common functions of unobtrusive observation is the collection of statistical information relevant to both the use and internal performance of the system. In the first category are such measures as the number of references to different files, the number of records per file, statistics based on the distribution of record lengths and on the number of records per file, the frequency with which different system commands are used, time profiles of system use, and a number of statistics based on vocabulary usage and the occurrence of various system failures. Command use and error message occurrence statistics in particular have been proposed [Katter, 1970] as a monitoring tool useful in formulating modifications to the user command set. Among the internal measures commonly collected unobtrusively in online systems are indexing and access statistics, along with a wide range of implementation-specific information useful in improving system efficiency. The online collection of system use and internal performance statistics has been found useful not only in improving system performance, but in planning for user training and documentation. In the SPIRES system, many of the sorts of online statistics discussed above are collected unobtrusively on a routine basis.

The kinds of information collected on an online basis vary considerably from one retrieval system to the next. Many researchers have felt it useful to record portions of the actual user dialogue in transcript form, allowing for subsequent analysis. Hillman [1968], for example, has suggested that a record be kept of each reformulation of a user request in the course of an interactive search. The SUPARTS system [Cook, 1970; Cook, et al. 1971] keeps a transcript of all user search input, along with a record of the number of documents found and the abstract and volume number of each document actually printed and additional identifying information. The resulting performance log is automatically summarized for easy reference. In the variation proposed by Rosenberg

[1973], such a record of search interactions is combined with a tape-recorded transcript of user comments collected throughout the course of search and calibrated according to actual elapsed time to reveal the plans, problems and reactions of the user at various points within the search process. Treu [1971] has suggested the use of human monitoring of, and even helpful intervention in, the search process. Ethical issues related to privacy which arise in considering the use of unobtrusive monitoring are discussed by Lancaster and Fayen [1973].

In conclusion, it should be emphasized that, despite the linguistic associations suggested by such terms as "relevance", "precision" and "recall", such constructs are typically operationalized using one of several simple evaluative procedures which may or may not capture the essential characteristics of a given type of information requirement. Even the phenomenologically oriented techniques that we have described may embody certain implicit assumptions about the process of retrieval which may not be applicable to a retrieval system whose purpose or design differs markedly from that of a traditional information retrieval system. Particular care must thus be exercised in applying the kinds of results that we have described to the kinds of retrieval tasks and systems with which our thesis research is concerned.

2.1.7 Emergence of the market for online retrieval

As we have seen, the essential elements of reference retrieval system design have their roots in classification and indexing mechanisms whose importance was recognized well in advance of the introduction of contemporary computer technology. With the increasing availability and decreasing cost of computer and communications technology during the past decade, however, the field of information retrieval has entered a dramatic new phase in which many of the applications which had earlier been forecast have indeed become realities.

According to a conservative estimate by King, et al. [1976], the provision of scientific and technical information to the private and public sectors, exclusive of state, local and city governments, and not including numerical data, accounted for at least $9 billion in annual revenues during the calendar year 1975. Growth in the market for scientific and technical information is apparently still proceeding at a better-than-linear rate, and has been projected to reach 6% of the gross national product by about 1985 [Nisenoff, Allen and Clayton, 1977]. Since scientific and technical information accounts for only part of the use of retrieval systems and services, these figures in fact understate the size of the emerging "information industry".

Although there are some applications—Selective Dissemination of Information (SDI), for example—for which batch processing still appears to offer certain advantages of efficiency, the most exciting area of growth in the availability of retrieval systems and services has without question been in the area of *online* reference retrieval. According to Williams [1978], citations for "the major portion" of scientific and technical literature published today can now be retrieved by computer. At present, over 55 million machine-readable citations, comprising more than 400 distinct bibliographic databases, are accessible for computer

searching. More than 75% of these citations may now be retrieved through commercially available online services, and many may be accessed through one of the several computer communication networks serving the North American and European user communities.

Of particular interest is the degree of vertical specialization which in the provision of retrieval-related products and services which has developed during the 1970's. Participants in this industry now include:

1. Database content producers and compilers
2. Implementors and vendors of retrieval systems
3. Providers of retrieval services using existing databases
4. Providers of retrieval services for use with customer-maintained databases
5. Data brokers serving as intermediaries between database producers and either end-users or the providers of retrieval services
6. Suppliers of communication network services for accessing databases and retrieval services from remote locations

While many firms in fact provide two or more of the above products and services to their customers, either internally or through the use of products or services provided by third-party vendors, the viability of of these activities as the basis for independently operated business concerns has now become apparent.

The need for another conceptually distinct service connected with the process of online retrieval is also becoming apparent: that of the professionally-trained searcher, responsible for translating the information needs of untrained end-users into effective search strategies and queries. A recent study conducted by the J. Walter Thompson agency on behalf of Euronet DIANE, as reported by Vernimb [1978], found that about 90% of all online searches under considera-

tion were in fact performed by "intermediaries specifically trained to use the terminal." 75% of these searches were carried out by librarians or information officers. While the need for such "linking agents" between the supplier and user of online reference retrieval services is now becoming evident, it remains unclear at present what class of individuals or groups is best suited to provide such services. A study by Firschein, Summit and Mick [1978], in which on-line retrieval services were made available to the public (graduate students, educators, technical professionals and librarians were in fact found to be the main users) through four Bay Area libraries, attempted to evaluate the public libraries as possible linking agents between online retrieval services and public end-users. The ultimate site of this critical function in the process of online retrieval, however, will probably not be known for some time.

While the structure of the market for online services continues to change rapidly, there is little doubt that such services are now playing an important and permanent role in a great many enterprises in the public and private sectors. Within the past decade, the provision of online retrieval services has grown from a largely experimental undertaking to a well-defined, and presently quite profitable, business. Among the largest vendors of online retrieval services are Lockheed Information Service (whose DIALOG system offers online access to over a hundred databases provided by a number of sources), System Development Corporation (offering similar services based on the ORBIT retrieval system), the National Library of Medicine, Bibliographic Retrieval Service, the Canada Institute for Scientific Information, and the European Space Agency. Last year (1978), the major online vendors conducted more than 1.5 million searches for their clients, and the number is certain to increase dramatically this year due to the rapid recent enlistment of new customers as the value of online retrieval systems has become more widely

appreciated. In the longer term, technological advances including video disks (allowing very inexpensive mass read-only storage), decreasing communications and computation costs and new peripherals technology are expected to further increase the size of the market for online retrieval services.

While the current and projected economic significance of online retrieval as an industry is of course of of interest in itself, there is a less obvious reason for our concern with the recent growth in this industry. Prior to the current decade, most of the work with information retrieval systems took place in an academic environment, where only limited experience with actual user requirements and behavior was gained. Although interesting data was in fact gained in the course of this research, the existence of a large "natural" community of users of publically available retrieval systems has made available a different and potentially valuable source of information about the necessary and useful features of such systems. As Martin [1974] notes, it may be possible by reviewing the sets of features incorporated in various retrieval systems in current public use to infer a great deal about the needs of users of such systems in general. Through the effects of competition in the marketplace during the recent past, Martin argues, it is likely that features, and on a broader level, systems, that fail to respond to actual user needs have tended to disappear, while features and systems that better satisfy such needs have tended to proliferate. In fact, there seems to have been a general convergence during the past decade or so toward a common set of features and capabilities (on at least a functional level, if not at the level of the detailed user interface) which may now be reasonably expected in most retrieval systems.

To be sure, there is a danger in this sort of inference. the great value (discussed later in this subsection) of standardizing the features supported by various retrieval systems might be expected to lead toward a convergence on

a single set of common features even at the cost of prematurely eliminating features which might in fact prove of value to groups of users whose needs fall outside the "mainstream" of retrieval system use. It is possible that this "positive feedback" effect might lead to the emergence of a growing class of users whose needs are well approximated by such "mainstream" features, and the exclusion from this user class (and thus from direct observation in the marketplace) of other individuals having divergent, but potentially satisfiable, information system needs. To the extent that this effect is in fact operative, inferences based on the performance of commercially available systems in the marketplace may thus lead to an overestimation of the commonality of user needs. At the same time, it seems clear that a great deal may been learned about the needs of a very large and economically significant class of users through the observation of those features commonly provided by currently successful information retrieval systems.

2.1.8 Features of contemporary online systems

A number of surveys and comparative analyses of retrieval systems have been conducted during the past fifteen years, each viewing the set of available systems from a somewhat different perspective. Welch [1968] conducted an early study of five interactive retrieval systems which was, however, considered out of date within a few years after its publication. Seidon [1970] outlined a set of features characterizing a number of bibliographic retrieval systems in the context of the requirements of the Biomedical Communication Network, and reviewed eleven then-operational systems for the presence or absence of these features. In the present decade, the National Bureau of Standards has played a prominent role in the analysis of available retrieval systems, reviewing the state of the art periodically in surveys published since 1971. In the NBS survey reported by Fong [1971], a list of system features is proposed for purposes of comparing document processing systems, and eight commercially available or government-owned systems available at the time of the report are reviewed with reference to this feature list. An NBS index compiled from questionnaire databased on 46 systems operational in 1973 and organized according to a somewhat different feature list was presented by Fife, et al. [1974]; other NBS Technical Notes [Marron, 1973; Marron, et al., 1974] are also concerned with the features of retrieval systems in the academic, government and commercial communities

Perhaps the most useful information regarding the essential features of contemporary retrieval systems, though, had its roots in a workshop held in January, 1971 under the sponsorship of the American Federation of Information Processing Societies (AFIPS) [Walker, 1971]. Representatives of different information retrieval systems discussed issues related to the user interface for interactive bibliographic searching, and most participants agreed to participate in a comparative analysis of the features of their respective systems. At a second meeting in November, 1971, sponsored by AFIPS and the American Society for Information Science (ASIS), a list of seventy features considered desirable in an interactive bibliographic search and retrieval system was compiled and a special interest group of ASIS for user online interaction (SIG/UOI) was formed to undertake a comparative analysis of the user interface of existing online systems. Subsequent funding from the National Science Foundation permitted a detailed comparison of eleven systems, based largely on a review of the users' manual for each system, half-day interviews with representatives of each of the systems, and a three-day workshop in April, 1973, attended by representatives of the eleven systems, a five-person expert panel, and several invited guests. The results of this study are presented in a report by Martin [1974], whose organisation will form the basis for our own in this subsection.

Following Martin [1974], we will divide our exposition of the features of contemporary reference retrieval systems three major categories:

1. Instructional, diagnostic and control features
2. Query formulation features
3. Result manipulation features

The second category may be viewed as the "heart" of the retrieval system, and although substantive differences will be observed in the manner in which different systems facilitate the process of query formulation by the user, nearly all draw upon the mechanisms discussed in Subsection 2.1.3. From the perspective of the user community, however, the logical characteristics of a system for query formulation and retrieval comprise only one of the determinants of its value in the context of a real set of information needs. Since most users

are not themselves experienced retrieval system designers, a system which in theory offers powerful command language capabilities but is found in practice to be impractically difficult to learn to use, unreasonably "unforgiving" in the face of user errors, or incapable of providing its results to the user in a useful form may well be of less value than a less powerful, but better user-engineered system. In this subsection we will thus attempt to take a broader view of the retrieval process, considering the actual environment in which ten of the eleven operational online systems compared by Martin [1974] are used. These systems are listed below, together with the organisations by which they are offered:

1. BASIS (Battelle)
2. DATA CENTRAL (Mead)
3. DIALOG (Lockheed)
4. INTREX (MIT)
5. NASIS (NASA)
6. ORBIT II (SDC)
7. RECON (Informatics)
8. RIQS (Northwestern)
9. SPIRES II (Stanford)
10. STAIRS (IBM)

To simplify our exposition, LEADER (Lehigh) will not be included in this discussion, since it differs substantially in its overall architecture from all of the other systems compared by Martin [1974], relying extensively on natural language processing capabilities and a mechanism for ranking results according to their presumed relevance, and deemphasizing the boolean operators which form an essential part of each of the other ten systems.

In describing the approach to instruction, error recovery and operational control adopted by contemporary retrieval systems, it is important to consider

not only the online support features provided by the system itself, but the printed documentation and human instruction ordinarily provided to new users as well. Different strategies have been adopted by proprietors of the ten systems under consideration for initiating the novice user. Although some of these systems (specifically, INTREX, SPIRES AND BASIS) may be used by individuals having received no prior instruction, most follow the practice that new users must first attend a short course outlining at least the basic techniques for online searching. Users of a few of the systems are generally given a small amount of personalized instruction during their first retrieval session. Users' guides and command summaries form an important part of the written documentation on which the novice and experienced user, respectively, can rely in the course of searching. Most of the systems provide either passive (tutorial) or active (user-selectable reference information) online training facilities intended primarily for users knowledgable in retrieval system behavior in general, but lacking specific knowledge of the features of the particular system being used.

Even the experienced user, however, experiences occasional memory lapses or makes certain mistakes in the course of searching, and each of the systems under consideration provides some facilities for reminding the user of correct command formats or recovering from common types of errors. Specific mechanisms include the ability to switch to a more detailed prompting mode, to request an explanation of the prompt or response just received, or to review the syntax and semantics of a particular command. All ten systems provide a telephone number that can be used as a last resort to reach a consultant for personal assistance, and several allow online communication with a consultant as well. STAIRS and INTREX also offer a traceback feature which reviews the history and intermediate results of a stepwise search procedure (discussed below), facilitating the detection of certain "global" search errors.

Other control features are included in each of the ten systems for purposes of limiting access to the retrieval system facilities at the network, computer, retrieval system or database level. Online monitoring (discussed earlier in this section) of various sorts, and the online solicitation of user comments are also supported by most systems for purposes of providing feedback to the system maintainers, thus facilitating the adaptation of the system to observed user needs and problems.

One significant practical problem confronting the designer of an online retrieval system whose importance has become apparent only through experience with actual systems having access to a large number of databases relates to the manner in which databases appropriate to a given information requirement are located and selected for use. (Several long-term research problems related to this issue are discussed in the next subsection.) Among those systems allowing access to more than one database (INTREX supports only a single database), initial selection of a database may be automatic (governed by a restriction or default selection associated with the particular user identification code) or user-selectable (prompted by an automatic or user-recalled list of available databases). Systems differ in the procedures for subsequent switching between databases, and may involve "backing up" to different points in the session.

One of the most valuable contributions of the Martin [1974] study was an analysis of the process of query formulation from the perspective of the novice user, who is likely to make certain sorts of systematic errors in formulating simple queries. Martin points out that the behavior of a user having no prior experience with online retrieval systems is often guided by one of two models for the process of information gathering borrowed from previous experience with non-computerised information processing. One model is that of the ordinary

question, as expressed in a natural language, and tends to lead to queries based on natural language syntax, thus containing many words which are in fact extraneous to the operation of a formal-language retrieval system. The second is that of a manual index (such as a library card catalog), and tends to result in queries containing only one or two salient concepts of the intended query, and foregoing the power of the combining constructs (discussed below) available in all of the retrieval systems under consideration. While we will not review in detail the effects of various system design decisions on the manner in which such common errors are accomodated, it should at least be mentioned that Martin has implicitly made a very strong case for importance of considering the implications of such decisions on the ease with which novice users are able to learn to use the system, rather than simply designing the user interface for a hypothetical "expert" user, as seems to have been the rule in earlier theoretical work in the field. Although this may not have been the designers' intent, certain systems indeed seem to be far more tolerant of queries typical of users who may not yet have an accurate model of the process of computer-based retrieval; this property seems to be quite important in practice.

The degree of commonality between the logical capacities of each of the ten systems under consideration for query formulation is quite surprising in view of the large number of conceivably useful mechanisms for document description discussed in Subsection 2.1.3. Whether this apparent consensus on the most practically significant mechanisms for query construction represents an accurate reflection of the needs of most users or is simply an artifact of an artificial convergence on an easily identifiable "lowest common denominator" for retrieval system behavior remains, in our estimation, an open question.

We will begin our discussion of query formulation by reviewing the "lowest level" (or, in general, the highest precedence) terms that may participate in

a query. Most systems provide some special mechanisms for dealing with numeric values, such as automatic unit conversion and the accomodation of ranges of values using relational operators (eg., "greater than", "less than or equal to"). Some systems also convert other special types of data, such as author names, into a standard format before beginning the search. In STAIRS, DATA CENTRAL and BASIS, predefined variants of some textual terms (abbreviations, for example) are treated as synonymous with the form specified by the user, and are included in the query as if explicitly specified using a disjunctive (*OR*) combining operator. String matching capabilities are provided by most systems, at least for purposes of suffix removal; the user typically specifies the desired stem followed by a special character or characters which is treated as matching any substring that might follow the stem in the citation records. In the example session with DIALOG presented in Subsection 2.1.10, for example, the terms COMPUTE? and AUTOMAT? are specified (set numbers 6 and 7, respectively) to effect retrieval of such variants as computer, computers, computed, automatic, automated, automation, etc. Several systems also allow the specification of an *embedded* "wild" character within a pattern word which is again treated as matching any substring occuring at the indicated point.

Most of the systems under consideration also allow the user to request a list of terms that are either alphabetically proximate or semantically related to a given word. In the former case, a small number of terms that alphabetically precede or follow the designated term are displayed along with the entry point term, which is highlighted in some systems. This feature may be used in much the same way as an ordinary dictionary, aiding the user who is uncertain as to the exact spelling of a term to scan the word list in the vicinity of the entry point to find the desired term, and may also be helpful in reminding the user

of certain suffix variants that were not initially considered. In some systems, the adjacent terms are assigned temporary reference identifiers so they can be quickly added to the query without typing the whole term. Several systems also provide a count of the number of records that each adjacent term would in fact retrieve, permitting an assessment of the best terms for inclusion in the search strategy. After displaying a fixed number of adjacent terms, most systems allow the user to ask for additional terms that precede or follow those already displayed.

The display of semantically related terms is based on the relations stored in an online thesaurus (which may be available only for some subset of the databases accessible through the system). Thesauri have been used for non-automated indexing and retrieval for some time prior to the availability of online retrieval systems, but deserve special mention in the context of contemporary interactive systems. Among the types of relationships that are represented in various online thesauri are synonomy (or near synonomy), abstraction (more specialized and more general terms, possibly at several "levels" in an "abstraction hierarchy) and a "catchall" category for other sorts of semantic relationships. The accessibility of relationships of these different types depends not only on the retrieval system, but on the organisation of the thesaurus for the particular database being searched. Systems differ with respect to the number of levels above and/or below the entry term that are displayed in response to a request for abstractively-related terms, whether related terms are displayed automatically or only upon explicit user request, and whether the types of relationships are displayed along with the list of related terms. In the case of synonomy or near-synonomy, certain systems used with certain thesauri also indicated the preferred term (as in an authority list) using a "Use (*term*)" or "Used For (*term*)" specification. As in the case of alphabetically ad-

jacent terms, some of the systems under consideration also indicate the number of records that would be retrieved if each related term were specified.

The following examples will serve to illustrate the structure of several thesauri actually accessible using existing online systems. The first is taken from the ABI/INFORM Controlled **Vocabulary** [Warner-Eddison, 1978], and includes only the "Use (*term*)" (abbreviated "U"), "Used For (*term*)" ("UF") and the "related in some other unspecified way" ("RT") relationships:

accomodation endorsement
RT *cosigning*
UF *accomodation paper*

accomodation paper
U *accomodation endorsement*

The next example, drawn from the Thesaurus of Psychological Index Terms [American Psychological Association, 1978] additionally illustrates the specification of a single level of abstraction in both the general and special directions:

academic achievement

Used For
achievement (academic)
gradepoint average
scholastic achievement
school achievement

Broader
achievement

Narrower
academic overachievement
academic underachievement
college academic achievement
mathematics achievement
reading achievement

Related
academic achievement motivation
academic achievement prediction
academic aptitude
academic failure
education
school learning

Finally, an example of the display of a portion of a multi-level abstraction hierarchy, adapted from the Thesaurus of ERIC Descriptors [Macmillan Information, 1977], is presented. More general terms are indicated by a number of colons (:) corresponding to the number of levels above the entry term in the abstraction hierarchy at which the related term is found, and more specialized terms are similarly designated using periods (.). In our example, the entry term *american history* is listed along with three successively more general terms (*history*, *social sciences*, and *sciences*), two distinct first-level specialisations (*mexican american history* and *united states history*) and one further specialization of *united states history* (*colonial history*):

:::*sciences*
::*social sciences*
:*history*
american history
.*mexican american history*
.*united states history*
..*colonial history*

Two systems (ORBIT and DIALOG) provide for the user-specified inclusion in the search query of *all* terms more specialized than the entry term. In STAIRS, the user may specify that two terms are to be treated as synonyms for purposes of retrieval.

Most systems provide several mechanisms for specifying that individual

words in a multi-word lowest-level term must be grouped together in one of several ways in any matching record. In various systems, words may be constrained to appear in a given order in the text, within a specified number of words or sentences of each other, within the same sentence, paragraph, field, index phrase or index linkage. Most systems also allow the user to specify that the words appear within a *particular* field (or index).

Other lowest-level term features represented by various systems include the specification of case characteristics (eg, upper case only, lower case only, first letter capitalized, or mixed case, most systems, however, do not distinguish between upper and lower case in search queries) and the centrality which the specified term must have in the target record (eg, MAJOR, SECONDARY, MINOR).

Turning now to the interconnection of lowest-level terms to form complete search queries, we find that most systems provide at least the boolean operators for conjunction (*AND*), disjunction (*OR*) and at least one form of binary negation (*AND NOT*, and in some cases *OR NOT*). All systems but INTREX combine operands according to a fixed priority scheme, with *AND* and *AND NOT* having higher priority than *OR* (and where present, *OR NOT*). In all ten systems, priority conflicts are resolved in a left-to-right order. Most systems allow the fixed precedence of operators to be overridden by parentheses. (It should be noted that the use of intermediate result sets, discussed below, also imposes implicit precedence re-ordering on the operands.)

Most systems allow queries to be formulated in a stepwise process, with intermediate search results collected and assigned a unique "set number". boolean connectives may then be used (and in DIALOG and BASIS, may *only* be used) to combine two or more intermediate result sets to expand, limit or otherwise refine the search result. In the sample DIALOG session in Subsection

2.1.10, for example, the terms *information retrieval* and *information systems* (set numbers 1 and 2, respectively) are combined using a disjunctive connective in line 3. Systems vary as to the exact procedure by which result sets may be combined, the manner in which the search process can be "backed up" to an earlier point and a new strategy adopted, the facilities available for reviewing the text of the query, and the mechanisms for saving the a request for use in the same session or in a future session and purging the result set so space may be reused for the collection of further result sets.

Finally, we turn our attention to the facilities available in the ten systems reviewed by Martin [1974] for manipulating search results. All of the systems provide mechanisms for both displaying search results at the user terminal and printing them off-line, allowing for both immediate feedback on the contents of the result set, which is often useful in reformulating a search request that yields unexpected or undesired results, and the provision of hard-copy listings of (potentially quite large) citation lists in a cost-effective manner. Some systems automatically display either one or several members of the result set, or list members continuously until interrupted by the user. Others require that the user explicitly request a display of results, in some cases following an appropriate system prompt. Retrieved records are displayed in a fixed order (typically either in the order the record was entered into the system or in reverse order of entry) in some systems, and in a database-dependent order in others. Some systems require that the user specify which fields are to be displayed, while others automatically display all, or a fixed subset, of the fields. The pattern terms which resulted in retrieval are highlighted in various manners in some of the systems.

In order to allow output meeting the specialized requirements of experienced users, most systems allow some measure of user control over the form of the

result display. Specific capabilities include the specification of fields to be displayed, the selection of alternate formats, either predefined or composed by the user through online interaction, and the specification of a sorted order in which the records are to be displayed, based on the contents of result fields or the degree of "goodness-of-match". Four of the systems under consideration also allow the computation of various statistics based on the values of specified result fields. During the process of display, some systems allow the specification of a new display format, and some allow a real-time decision as to whether each record is to be retained in the result set or flushed as irrelevant. Various options for displaying the actual microform text referenced by a result citation, generating graphs and/or statistical analyses based on the retrieved records and specifying an address to which printed results are to be mailed are also provided by some of the systems.

2.1.9 Recent trends in retrieval systems research

During the past decade, the focus of research activity in the field of information retrieval has shifted significantly in several respects. While we will make no attempt at comprehensiveness, this subsection will attempt to convey a feeling for a few of the most important research problems that have attracted increased attention in recent years, and perhaps more importantly, for certain changes in the generally perceived organization of the field itself.

Several trends are apparent in reviewing the literature of reference retrieval during the past few years. The most obvious is an increasing concentration on online, interactive systems, as opposed to batch retrieval. As mentioned in the previous subsection, there are still tasks of practical significance (selective dissemination of information, for example) for which batch processing algorithms, often qualitatively different from those that must be employed in an interactive framework, continue to offer significant advantages in efficiency over online alternatives. While description-matching in batch mode may thus be more cost-effective in certain applications, such factors as the provision of immediate feedback in the form of intermediate search results, access to online thesauri and other aids to the process of query formulation, and the elimination of delays in obtaining the the results of a search have increasingly tended to focus research attention on the problems of interactive search.

This increasing interest in interactive retrieval has been marked by the introduction of several new forums for the exchange of information about online systems. *Online Review*, a journal publishing articles dealing specifically with interactive information retrieval systems, began publication in 1977, and, according to a study by the J. Walter Thompson agency [Thompson, 1978], already serves as a primary information source to users of such systems. Other

important forums for information about interactive retrieval include *Online Magazine*, various conferences organized by Online Services, and users groups composed of customers of different online retrieval services. The proportion of papers published in the *Journal of the American Society for Information Science* (ASIS) which deal with the problems of online retrieval has steadily increased over the last several years, and increasing emphasis on these problems can be observed in successive volumes of the ASIS-published *Annual Review of Information Science and Technology*. A quick review of the contents of the journal *Information Processing and Management* (called *Information Storage and Retrieval* until 1974) tends to confirm this trend.

A second change that may be observed in the literature of reference retrieval involves a deemphasis of the very general class of quantitative performance measures discussed in Subsection 2.1.5 (various recall and precision measures, for example) in favor of what might be characterised as a concern with the more "practical" problems of retrieval using "typical" systems of the sort described in Subsection 2.1.8. In part, this change has resulted from explicit consideration of some of the intrinsic weaknesses of early formalizations of these notions of retrieval system performance. Wilson [1973], for example, points out that classical relevance measures fail to capture the notion of the "situational" relevance of retrieved information -- the extent to which such information answers "questions of concern" to the searcher, given his current knowledge state and goals in gathering information. The more realistic notions of relevance and precision suggested by such observations, however, have thus far eluded explication at the level of concreteness typical of the classical measures introduced in Subsection 2.1.5. An equally important influence on the prevalent concerns of retrieval system design and evaluation, though, is the convergence noted in the previous subsection on a more tightly constrained

set of online retrieval system features than could be anticipated at the time the classical performance measures were formulated. The increasing functional commonality among different retrieval systems has permitted the formulation of evaluation criteria specific to the sorts of features actually implemented in typical online systems.

At a broader level, the emergence of a reasonably well-defined set of common retrieval system functions has made possible a clearer separation between "mainstream" retrieval systems research and the body of research aimed at more ambitious long-term goals. the former category of research is becoming increasingly "driven" by problems arising through experience with operational online systems. Illustrative of the latter category is the ongoing theoretical work in automatic indexing, which has recently begun to deal with very difficult semantic and pragmatic (as opposed to strictly statistical, lexical and syntactic) issues in natural language comprehension.

A third major trend that is apparent in the pattern of research activity in the field is a growing emphasis on the problems of online access to a large number of databases that may differ in their structure and in the support they provide for various retrieval tools, and on the differences at the user interface level between various online retrieval systems. Two major approaches to these problems have received the bulk of the attention of researchers in these areas. The first is exemplified by efforts at MIT [Heintjes and Marcus, 1974] to develop a common language that might be used to translate between the protocols of different systems sharing a common functional core of the sort described in the previous subsection. The second approach involves the proposed introduction of standards at various levels (see Williams [1974] for a general discussion) for purposes of minimising the differences confronting the user of different retrieval systems and databases. Several authors [eg, Martin, 1974, Atherton, 1978]

have stressed the need for a common vocabulary for talking about searching as a first step in the development of standards for the systems themselves. Atherton [1978] has in fact proposed that a subcommittee of the American National Standards Institute (ANSI) be formed to consider the adoption of standards for the "technical terms and symbols" used in each of the following four functional areas of online retrieval systems (derived from the consensus of participants in the January, 1971 "User Interface" meeting [Walker, 1971] in Palo Alto discussed earlier):

1. Search/query negotiation
2. Instructional/tutorial
3. Output
4. Supervisory/operational control

In the area of search term vocabularies, considerable disagreement remains regarding the ultimate role of standardization. At one end of the spectrum are the supporters (Artandi [1972], for example) of strict vocabulary control in indexing and retrieval. Others (eg, Lancaster and Fayen [1973]) dismiss such control as impractical, and instead hold hopes for such research advances as the development of powerful natural language interface subsystems, which could result in systems insensitive to the sorts of descriptive variation now found among different systems. In addition to the goals of assuring compatibility and convertibility among different retrieval systems, researchers investigating the problems of vocabulary standardization must consider the issue of *timeliness* of vocabularies—the extent to which a given vocabulary remains useful over a period of time in characterizing a changing set of content areas. Harris [1974] provides an excellent review of many of the salient issues in vocabulary standardization.

2.1.10 An example session with DIALOG

The following is a transcript of an actual session with Lockheed's DIALOG system:

```
ENTER YOUR DIALOG PASSWORD
xxxxx  LOGON File1 Thu 12APR79 18:36:08


ECONOMIC ABSTRACTS INTL NOW ONLINE.  NATL
NEWSPAP INDEX RESCHED MAY 1.  SEE ?NEWS
  8 STEPS INFORMATION RETRIEVAL OR INFORMATION SYSTEMS;
  8 ON LIN SYSTEMS
    1   2160 INFORMATION RETRIEVAL
    2   2684 INFORMATION SYSTEMS
    3   4076  1 OR 2
    4      0 ON LIN SYSTEMS
  8 ON LINE SYSTEMS; 8 STEPS COMPUTE? OR AUTOMAT?;
  8 LIBRARY REFERENCE SERVICES
    5    562 ON LINE SYSTEMS     COMPUTER SYSTEMS IN WHICH PERIPH
    6  13180 COMPUTE?
    7   3047 AUTOMAT?
    8  14865  6 OR 7
    9    563 LIBRARY REFERENCE SERVICES
```

S STEPS BIBLIOGRAPHIC CITATIONS OR BIBLIOGRAPHIES OR CITATION INDEXES

10 651 BIBLIOGRAPHIC CITATIONS OR BIBLIOGRAPHIES OR CITATION INDEXES

11 12489 BIBLIOGRAPHIES

12 519 CITATION

13 13157 10 OR 11 OR 12

C 5 ABD 13

C 5 ABD 13

Invalid set-range operator

C 5 AND 13

14 44 5 AND 13

T 14/6/1-10

14/6/1

EJ187159

Magazine Index: Popular Literature Online

14/6/2

EJ187066

Online Information Retrieval Bibliography

14/6/3

EJ187064

Use of Online Bibliographic Search in Public Libraries: A Retrospective Evaluation

14/6/4

EJ168227

An SDC Based On Line Search Service: A Patron Evaluation Survey and Applications

14/6/5

EJ114622

Searching the Medlars Citation File On-Line Using ELHILL 2 and STAIRS: A Comparison

14/6/6

EJ084869

The Cost-Performance of an On-Line, Free-Text Bibliographic Retrieval System

14/6/7

EJ076802

Automated Cataloging and Reclassification by ATS

14/6/8

EJ054837

Networks with Emphasis on Planning an Online Bibliographic Access System

14/6/9

ED157538

Promotion and Publicity for On-Line Bibliographic Services in Academic Libraries.

14/6/10

ED157530

The Use of OCLC Terminals in the Public Service Areas of Eight Ohio Libraries.

LIMIT 14/066621-999999/ED; LIMIT 14/062752-999999/EJ

15 30 14/066621-999999/ED

16 7 14/062752-999999/EJ

LIMIT 14/110595-999999/ED; LIMIT 14/121927-999999/EJ

17 17 14/110595-999999/ED

18 4 14/121927-999999/EJ

C 17 OR 18

19 21 17 OR 18

PR 19/5/1-21

Printed19/5/-21

LOGOFF

12apr79 18:44:42 User8862

$3.60 0 144 Hrs File1* 9 Descriptors

$2.10 21 Prints

$5.70 Estimated Total Cost

LOGOFF 18:44:49

2.2 Motivation for Knowledge-Based Retrieval

In Section 1.1, we defined conceptual matching as "the assignment of a correspondence between entities that humans perceive as being highly and systematically structured, based on selected significant characteristics of those structures." In order to identify the important respects in which our knowledge-based information retrieval system differs from the kinds of commonly used retrieval systems reviewed in the previous section, it will be helpful to consider the special properties of such systematically structured entities which distinguish distinguish a matching task as "conceptual" in nature. After introducing these properties in Subsection 2.2.1, the abstract descriptive capabilities of traditional and conventional information retrieval systems are considered in Subsection 2.2.2. Within this framework, the distinctive goals of a conceptual matching system are introduced in Subsection 2.2.3.

2.2.1 Essential characteristics of structured entities

The most interesting aspects of conceptual matching are related to the fact that both the matchable entities and the match criteria exhibit a rich conceptual structure. It is the perception of this conceptual structuring which provides the basis for a meaning-based evaluation of the appropriateness of potential pairings. In this subsection, we will outline those distinguishing aspects of structured entities that lend a conceptual flavor to the class of matching tasks under consideration

In the course of this discussion, the notion of *structural level*, which reflects the scope and grain of a particular observation of some structured entity, will be introduced. The range of possible levels of perceived detail will be conceived of as a *structural hierarchy* in which a distinctive set of emergent *properties* are made evident at each higher level through the imposition of *constraints* on the configuration of lower-level elements. Loosely speaking, a system may be considered structured whenever it can be fruitfully examined at more than one level of conceptual detail, with different properties of the system becoming salient to the observer depending on the chosen "grain" of observation.

An automobile company, for example, may be observed at any of several levels of detail, each associated with different sorts of characteristics. Viewed at a detailed structural level, the activities of each automotive worker may be described in terms of such basic physical activities as hand and arm movements, the production of phonemic strings, and primitive acts of personal locomotion. With the addition of a detailed mechanical description of each machine found in each manufacturing plant, one might imagine the possibility of providing a very accurate description of the detailed functioning of the firm over some period of time.

Such a description, though, would have little value in coherently expressing those aspects of these operations that would be considered most important to a productive business enterprise. While a detailed analysis at this level would be essential for the solution of certain problems, it is unlikely that a consumer comparing the advantages of various makes and models, for example, or an investor considering the purchase of stock in the company, would be interested in considering the detailed behavior of each machine and employee in the firm.

Ascending the structural hierarchy to a level of detail corresponding to the corporation as a whole, though, a very different set of characteristics *emerges* from the same physical entity. At this new structural level, such features as a product, a board of directors, and such abstract notions as capital assets are observed. Each of these is in fact made up of components observed at lower levels, but the observation of *characteristic* elements and inter-element relationships makes it possible to view these same phenomena at a "higher" structural level which is likely to be of greater use to an investor. A historical analysis of industrial production during the Second World War or an investigation of current unemployment patterns would probably involve yet a higher structural level corresponding to the automobile *industry* taken as an aggregated whole. The level in the structural hierarchy that is chosen for observation is thus highly dependent on the particular purpose of that observation.

It is significant that not *all* possible interacting combinations of people, equipment and actions can be usefully observed in terms of the characteristic features of automobile firms. Rather, it is the introduction of a characteristic set of **constraints** on the detailed configuration of this system which leads to the **emergence** of conceptually useful properties at the corporate level of the structural hierarchy. Most important, the perception of "similarly" constrained

systems may allow an observer to profitably apply a common interpretive framework to related *sets* of configurations. When the potential investor asks his broker about the management of a particular automobile firm, for example, the broker need *not* explain the function of a corporate president in terms of low-level human behavioral primitives. Perceived commonalities in the sorts of constraints observed in *all* corporations allow the broker to assume that his client is generally familiar with such emergent elements as the general role of a corporate president.

Emergent properties, then, may be associated with similarly constrained *sets* of lower-level configurations. Clearly, though, a given configuration may be regarded as a member of any number of conceivable configuration sets depending on the particular constraints chosen for use in the partitioning process. The set of properties that emerges with a transition to a higher structural level is thus dependent on the particular set of configurations whose characteristic constraints are considered "similar" in the context of current requirements. One degree of freedom is related to the *size* of the set of conceivable systems that are grouped together for purposes of factoring out common emergent properties. At a given structural level, certain emergent characteristics may be shared by a large number of particular detailed configurations, while others may be applicable only to some proper subset of these configurations.

Some of the characteristics of the above example entity that emerge at the level of the automobile firm, for instance, are in fact features of organisations in general. (Most organisations can be assumed to have members, for example, along with some form of leadership and a purpose for existence). Other characteristics which emerge at the same structural level are particular only to corporations—the existence of stockholders, products and a board of directors, for example. When the same system is viewed as a member of another set of

configurations whose detailed components are even more tightly constrained, a third set of properties, relevant only to automobile corporations, emerges. Furthermore, different properties may emerge at this structural level depending on the particular subset that is chosen. A political party, for example, may, like a corporation, be viewed as a specific kind of organization. While lacking stockholders, products and directors, this specialized type of organization reveals such emergent properties as candidates, conventions and Central Committees.

Furthermore, most configurations may be usefully regarded as members of several non-nested configuration sets, each associated with its own set of emergent characteristics. Beer, for example, may be viewed as a special case of either a beverage or an intoxicant, depending on the characteristics salient to the particular conceptual task at hand. Not all beverages qualify as intoxicants, nor are all intoxicants ingested in liquid form, but a number of the salient properties of beer are in fact general characteristics of beverages or intoxicants in general. The salience of alternative sets of constraints thus introduces further possibilities for the choice of properties to be considered emergent at a given structural level.

In summary, the particular set of characteristics that emerge from a different observational viewpoint is dependent not only on the level of detail chosen for observation, but on the set of "similar" configurations of which the observed system is considered a member. It may thus be useful to consider a second parameter, distinct from the choice of a structural level, as a determinant of the particular emergent interpretive framework which is salient in a given situation. This parameter, reflecting the *extent* and *dimension* of structural constraint, and influencing the specificity and selection of emergent properties, is the basis of the notion of *abstraction*. As in the case of structural

detail, it will sometimes be meaningful to speak of abstractive *levels* ranging from specialized to general, each denoting a progressively less restrictive set of structural constraints.

The "factoring" of emergent characteristics at appropriate abstractive as well as *structural* levels is critical to the task of conceptual classification and retrieval. Part of the significance of the notion of abstractive levels in naturally and artificially intelligent systems is based on on the *inheritance* by specialised configuration sets of the emergent properties of a more general abstractive "parent". Certain properties of an abstractive "child" are specific to the more tightly constrained system, and are unrelated to any properties of the parent. The notion of a product, for example, is meaningful in the context of a corporation, but has no analogue in the more abstract case of a general organisation. The officers and directors of a corporation, on the other hand, may be viewed as specializations of the notion of leadership which were listed as one of the important features of organisations in general.

2.2.2 Descriptive capabilities of conventional systems

In this subsection, we will consider some of the limitations of past and present information retrieval with respect to the goals addressed by conceptual matching in general and knowledge-based information retrieval in particular. These limitations are best expressed in terms of the notions of structure and abstraction introduced in the previous subsection, which provide a unifying basis for the examination of existing retrieval schemes and for the justification of our proposed approach to the problem of meaning-based retrieval.

In all fairness, it is worth noting that various specialised and functionally limited instances of most of the mechanisms on which our approach is based can be found even among systems predating the introduction of contemporary "generalised" computer-based information retrieval systems. While the specific retrieval systems introduced in Section 2.1 have often employed various combinations of the mechanisms described below, we have attempted to identify several conceptually distinct categories of retrieval tools that will prove useful in discussing the fundamental import of our knowledge-based approach to retrieval.

Simple Abstraction

The simplest abstractive mechanism for document classification involves the assignment of each of a set of target documents to one of a fixed, smaller set of categories on the basis of shared characteristics. The value of simple abstraction lies in the fact that the user is often able to first identify potentially relevant categories and limit search to the documents in these subsets of the target collection. It is difficult to find a document retrieval system that does not use the abstractive mechanism in some form.

Abstraction Hierarchies

In large collections, the use of a single level of classificatory abstraction leads either to a great proliferation of document classes or to the existence of classes having so many members that search within each category becomes difficult. In an abstraction hierarchy, on the other hand, "top-level" classes are in turn subdivided into further categories, and so on, such that the user can first identify relevant classes, then further restrict his search to appropriate subclasses of these classes, continuing until a manageably small set of documents remains. Library catalog numbers such as the Library of Congress and Dewey Decimal classifications exemplify the use of hierarchical abstractive mechanisms.

Multiple Abstraction (Independent Feature Mechanisms)

Multiple abstractive mechanisms involve the characterisation of each target document as a member of *several* non-nested categories, each corresponding to a set of characteristics shared by its members. Multiple abstraction allows *independent* features of the target to be separately noted and utilized without an exponential increase in the number of categories that would be required if each possible combination of features were assigned a distinct simple abstractive category. A single paper might, for example, be classified as a *Document-dealing-with-automata-theory* and a *Monograph*, and the two characteristics used separately for various retrieval tasks. Examples of multiple abstractive mechanisms include indexing systems in which the targets are described by cards with punched edge slots (see Section 2.1), each slot representing a single feature.

Schematic Structures (Attribute-Value Mechanisms)

The use of schematic mechanisms in document retrieval is based on the observation of a small set of common characteristic attributes that may have different values in the case of different target documents. This observation suggests the use of a *schema* for documents in general which may be *instantiated* by filling various of its *slots* so as to specify the particular values of the corresponding attributes of each target document. Library card catalogs, for example, make use of a schematic mechanism in which such slots as *author*, *subject* and *number of pages* are filled with the particular values associated with each document in the collection. Schematic systems thus differ from the multiple abstractive mechanisms discussed above in that the specification of each feature involves two elements: a particular slot, and its corresponding filler.

Schematic structures permit the user (or the retrieval system) to distinguish between a book by Kennedy about Johnson, for example, and one by Johnson about Kennedy. To be sure, this distinction could be made using a multiple abstractive classificatory scheme through the creation of separate features for *Author-Kennedy*, *Subject-Johnson*, *Author-Johnson* and *Subject-Kennedy*. Through the use of schematic descriptive strategies, though, it is possible to "factor" relevant information and procedures together in association with the appropriate slots. Special rules for standardizing publication date formats and retrieving documents published within a specified time period, for example, may be specified once in association with the *publication-date* slot, and not repeated for each of the multiple abstractive categories *Publication-date-1936*, *Publication-date-1937*, etc. Similarly, information most naturally associated with a given filler (e.g., Johnson) may be specified once, independent of the slot for which the entity in question will serve as a filler.

We now turn our attention to the use of such traditional descriptive mechanisms within the typical contemporary information retrieval systems. In Section 2.1, a number of "conventional" retrieval systems, utilizing vastly different mechanisms for description and matching, were considered. In order to simplify our exposition of the relevant limitations of such existing systems, however, it will be useful to restrict our attention to a simplified model of a typical "state-of-the-art" data base system such as SPIRES [Parker, 1970], which is currently used for a number of practical document retrieval tasks. For purposes of distinguishing the essential aspects of our proposed research, the basic functional operation of a SPIRES-like system can be summarized as follows:

1. *Schema Selection*. A single type of schema is first chosen for use in the retrieval session through the user selection of a particular set of targets (a "subfile" of homogeneous "goal records", in SPIRES), each of which is represented by an instantiated target schema. This choice may be thought of as a single abstractive narrowing which specializes the search to the target class in question.

2. *Instantiation of the Request Schema*. The user next instantiates (partially or completely) a request schema having the same form as the target schemata, by specifying for selected slots either particular fillers, simple predefined constraints (typically involving arithmetic or string comparisons) or boolean combinations of particular and constraining specifications. This process may be regarded as a one-level process of structural elaboration.

3. *Retrieval of Matching Targets*. The system now applies a simple matching algorithm to decide which targets match the instantiated schema. A request schema is considered to match a given target schema (of the same type) when the boolean combination of particular and constraining specifications currently filling each of its slots is consistent with the corresponding target slot. Such consistency may involve either simple equality tests or specially defined type-specific procedures associated with the slot in question; unfilled request slots are always deemed to match the corresponding target slot.

This summary of course ignores many substantive characteristics of a SPIRES-like system which were discussed in Section 2.1; the above analysis, however, should be adequate to highlight the most important respects in which the goals and capabilities of our knowledge-based information retrieval system differ from those of contemporary systems.

2.2.3 Requirements of a conceptual matching system

As noted in Section 1.2, one of the most important characteristics of the sort of system that we have implemented is the ability to retrieve highly specific targets from among a large class of conceptually heterogeneous documents. It is our contention that the sorts of systems discussed in the previous subsection, which use only a single, fixed schematic structure for describing the user's request, are seriously and intrinsically limited in this regard. Such systems can of course be immensely useful in either of the following situations:

1. *Highly Restricted Target Collection.* The target collection is restricted to a set of documents which share most of the specific attributes that will be incorporated in user requests. A graduate student working in the area of toxicology, for example, might well make use of such a system if the target class were restricted to papers on the mechanisms of poisons and their antidotes.

2. *Very General Classification Criteria.* A manageably small set of attributes general enough to apply to most of the target documents (*author* and *subject*, for example) is found sufficient to adequately distinguish the targets for purposes of the application at hand.

In the first situation, though, the original task confronting the user is merely replaced by the new conceptual matching problem of finding the relevant collection. If each collection is specialized to the degree necessary to achieve the benefits of Situation 1, the number of such collections will be so large as to make the problem of schema selection itself a formidable task, particularly in the common case where the desired target may be regarded as spanning disciplinary boundaries. In the case of Situation 2, on the other hand, the advantage of a large and heterogeneous document collection must be balanced against severe limitations on the capacity for discrimination of relevant documents on the basis of criteria specific to highly specialised targets.

Our work is directed toward the discrimination of appropriate targets on the basis of very specific properties: those emerging at highly specialized abstractive levels and highly detailed structural levels—without the sacrifice of heterogeneity in the target collection. We should like it to be possible, for example, to retrieve a target document whose description might be expressed in English as "A systems reference manual for a PASCAL cross-compiler running under the NOS operating system on a CDC Cyber 175 computer and generating target programs for execution on a DEC PDP-11 computer running RSX-11". Furthermore, this target should be distinguished from, say, a manual for a similar compiler in which the host and target machines were reversed. In the remainder of this subsection, certain critical limitations of existing systems in attaining this sort of performance will be examined, and the most important departures embodied in our work will be outlined. We begin by discussing some of the issues related to *abstraction* (see Subsection 2.2.1) to which our approach to document description is directed.

In conventional document retrieval systems having the capabilities discussed in Subsection 2.2.2, each document being catalogued and each request being specified is implicitly viewed as a member of a single abstractive class characterised by the structure of the fixed schema associated with the target collection. In considering the problems of retrieving actual documents according to the kinds of conceptual criteria described above, though, it becomes evident that both documents and requests (along with their structural constituents, as will be seen shortly) must often be viewed as members of *various* abstractive classes, including both nested and intersecting categories.

The need for descriptive information associated with different abstractive *levels* (that is, with mutually nested abstractive categories) is a direct consequence of the simultaneous requirements of heterogeneity and specificity

discussed above. A request for an atlas published in 1935 and showing the political boundaries of Europe, for example, might involve the filling of a *publication-date* request slot, factored in association with a general document schema, along with *geographic-area* and *map-type* slots associated with atlases in particular, and not generally meaningful for other sorts of documents. In the course of a cataloguing or description-building session, it will become possible to gradually specialise the target or request schema, respectively (again along with its emerging substructures), often adding new descriptive details at each successively more specialised abstractive level. In addition to hierarchical abstractive mechanisms, the capability for *multiple* abstraction appears quite essential for a knowledge-based retrieval system. The utility of multiple descriptive mechanisms is based closely on the sorts of considerations already discussed in Subsection 2.2.2.

Let us now consider some of the *structural* (in the sense of Subsection 2.2.1) issues raised by our approach to knowledge-based retrieval. Request and target descriptions in current SPIRES-like systems incorporate a single level of structure defined by the document schema and its associated set of slots. While the provision of one level of detail is sufficient for the performance of many practical retrieval tasks, this restriction imposes serious limitations in cases where one or more of the slot fillers (which themselves typically represent entities having complex systemic structure) must be described in a structured manner in order to distinguish appropriate targets.

Consider the paper "An Application of Heuristic Search Methods to Edge and Contour Detection", which was in fact published in the December, 1976 issue of the *Communications of the Association for Computing Machinery*. Given a sufficiently sparse target collection, a single-level structural description in which the two abstractive descriptors *Artificial-intelligence* and *Graphics-*

and-image-processing (actual ACM subject categories) were used to fill the top-level *subject* slot might well be sufficient. In order to distinguish the subject area of this article from a paper describing, for example, Mitch Model's thesis research on graphic display tools for monitoring the operation of knowledge-based systems (without introducing a proliferation of unreasonably idiosyncratic subject categories), on the other hand, a second level of structure is strongly needed.

Given a second, widely applicable schema representing the general notion of an *Application* and having slots for the *tool* and *task*, the subject of the *CACM* paper could be described by filling the *tool* slot with *Artificial-intelligence* and the *task* slot with *Graphics-and-image-processing*. In the case of Model's work, the *tool* would be classified as *Graphics-and-image-processing* and the *task* as *Artificial-intelligence*. To describe the papers themselves, the top-level *subject* slot would itself be filled with a *substructure*: an appropriately instantiated *Application* schema. Because of the need for highly specific discrimination at multiple structural levels, the descriptions used in a conceptual retrieval system will often have the form of a "network" of schematic forms, interlocked through the slot/filler relationship.

It is the many-to-many mapping between slots and their potential fillers which lends combinational power to a schematic mechanism supporting multiple structural levels. On the one hand, each slot may be filled with any of a number of alternative schemata, each having a distinct set of slots. If a schema for *Election* were to replace *Application* as the filler of the *subject* slot, for example, an entirely new set of slots would be introduced in place of *tool* and *task*. The provision of multiple levels of schematic structure thus allows the use of specialization to introduce highly specific attributes associated not only with the top-level (document) description, but with the description

of schematic slot fillers of the top level schema, for the fillers of those fillers, and so on as may be necessary for target discrimination. Conversely, a given schema may serve as the filler of any of a number of slots (possibly at various structural levels) as illustrated in the *CACM* example above.

The major departures of the knowledge-based descriptive language introduced in the following section from those used in most conventional information retrieval systems are thus related to:

1. the use of multiple levels of schematic structure, each having slots that can be filled by instances of schemata having various alternative structures.

2. the availability at any of these levels of descriptive elements having various degrees of specificity, and the possibility of using these descriptive elements either singly or in combination to describe a given entity from alternative viewpoints.

While these characteristics impose a requirement for the maintenance of domain-specific knowledge, the resulting capabilities extend far beyond those of existing information retrieval systems, and thus merit serious consideration.

2.3 The Knowledge-Based Description Language

The previous section suggested several capabilities that would appear to be quite useful in a description language for the task of knowledge-based information retrieval. The knowledge-based description language adopted for use in our demonstration system, which is based closely on the knowledge representation language *KRL* [Bobrow and Winograd, 1977], supports each of these requirements in a natural fashion. The description language, along with the rules for description matching, will be introduced in Subsection 2.3.1. In the second subsection, the manner in which descriptions are employed in the specification of domain-specific relationships in the knowledge base is described.

2.3.1 Descriptions and descriptors

Rather than begin with a formal treatment of the knowledge-based description language that was used in constructing our thesis system, we have chosen to first illustrate the essential components of its descriptions by examining a simple (albeit rather contrived) example of a hypothetical document description:

```
a Document
  authors
      set-with-all-of
        Thompson
        Walters
      set-of
        an Engineer
  countries-of-publication
      set-with-any-of
        USA
        Great-Britain
  subject
      involves
        an Invention with
          purpose
              or
                Power-generation
                Power-transmission
  copyright-dates
      set-with-exactly
        1935
        1962
a Survey-Article
```

(In the actual description language, parentheses are used extensively for purposes of grouping; we have taken the liberty of omitting the parentheses in this and most other examples, using indentation to convey the same structural information.)

The above *description* is made up of two *descriptors*—one of which (which in fact includes the great bulk of the top-level description) characterizes its referent as a *Document*, and the other more specifically as a *Survey-article*. The use of descriptions composed of multiple descriptors, each reflecting a different way of viewing the same real-world entity, gives our descriptive language the power of multiple abstraction, whose practical importance was discussed in Section 2.2.

In all, there are eight types of descriptors that may be included in the descriptions of our language, each of which is illustrated in the above example. The two that we have just mentioned are called *perspectives*, which are the schematic descriptive mechanisms (Section 2.2) of our language. Each perspective includes a single *prototype* (*Document* and *Survey-article*, in the examples under consideration) with which is associated a particular set of characteristic *slots*. One or more of these slots may be filled with another description in any particular instance of a perspective, thus providing the *structural* descriptive capabilities discussed in Section 2.2. In our example, the *authors*, *countries-of-publication* and *subject* slots are each filled. Alternately, a perspective may appear without any of its slots filled, as in the case of the *Survey-article* descriptor. A target perspective is deemed to "syntactically" match (see Subsection 2.4.7) a pattern perspective exactly in the (recursively defined) case where the prototypes are identical, and where every slot which is filled in the pattern has a matching description in the corresponding position in the target.

Several other types of descriptors are embedded within the slots of our top-level example description. The simplest of these are the *individual* descriptors *Thompson*, *Walters*, *USA* and *Great Britain*. Individuals are the only "atomic"

(i.e., non-decomposable) descriptor type, and may, to a first approximation, be thought of as corresponding to single, specific entities in the "real world". Individual descriptors in the pattern description are matched only by identical individual descriptors in the corresponding target descriptions.

The semantics of the four "set types" -- *set of*, *set with exactly*, *set with all of* and *set with any of* -- conform reasonably well to the meanings suggested by their names. In our example, the authors are described as a set whose members are all engineers, and which includes both Thompson and Walters in particular (possibly along with others). The set of countries in which the hypothetical document is published is required to include either the USA, Great Britain or both. Finally, the document is to bear exactly two copyright dates: 1935 and 1962. For the most part, the rules for matching descriptors of the various set types are fairly intuitive; for details, the reader is referred to the matching axioms themselves (Appendix), which should be reasonably easy to understand after completion of Section 2.4.

Involves descriptors support description-matching on the basis of *structural embedding*. Specifically, an Involves descriptor in the pattern which has a description D as its argument will successfully match against any description in the corresponding position within the target which either itself matches D or has some *subdescription* which matches D. In intuitive terms, description D_1 is called a subdescription of description D_2 if D_1 is lexically nested at any depth within the body of D_2, either as a slot filler (in the case of a perspective) or an argument (in the case of a set type or disjunctive descriptor). For a more precise definition, the reader is again referred to the matching axioms.

Finally, the *disjunctive* descriptor is defined to match against a given target descriptor if *either* of its two arguments (the individuals *Power generation* and *Power transmission*, in our example) would match against that target descriptor.

Figure 2.1 defines the precise syntax of our knowledge-based description language. Consistent with the usual conventions, the names of nonterminals are enclosed in angle brackets, with alternative choices separated by vertical bars. Braces are used as meta-symbols for purposes of grouping; parentheses, on the other hand, are symbols of the description language itself. A "plus" superscript is used to indicate that a syntactic element may occur one or more times, while a superscripted asterisk marks an element which may occur zero or more times.

(*description*) :: = ((*descriptor*) *)

(*descriptor*) :: =
(*individual*)
(*perspective*) |
(*set of descriptor*) |
(*set-with-exactly descriptor*) |
(*set-with all of descriptor*) |
(*set-with any-of descriptor*) |
(*involves descriptor*) |
(*disjunctive descriptor*)

(*individual*) :: = (*LISP identifier*)

(*perspective*) :: = ((a | an) (*prototype*) (*filler pair*) *)

(*prototype*) :: = (*LISP identifier*)

(*filler pair*) :: = ((slot name) (*description*))*

(*set-of descriptor*) :: = (set-of (*description*))

(*set-with-exactly descriptor*) :: = (set-with-exactly (*description*) +)

(*set-with-all-of descriptor*) :: = (set-with-all-of (*description*) +)

(*set-with-any-of descriptor*) :: = (set-with-any-of (*description*) +)

(*involves descriptor*) :: = (involves (*description*))

(*disjunctive descriptor*) :: = (or (*description*) +)

Figure 2.1 Syntax of the Knowledge-Based Description Language

2.3.2 Antecedent-consequent pairs

In the course of examining a number of actual documents chosen from the domain of computer science, we were able to identify several kinds of deductive inference mechanisms that might well prove useful in a working knowledge-based retrieval system. In our actual thesis system, however, only a single, relatively simple form of inference--based on *antecedent-consequent* (or simply *AC*) *pairs*--was chosen for purposes of demonstrating our approach to meaning-based matching. Each antecedent consequent rule expresses a relationship between two descriptions--the *antecedent* and the *consequent*—which may be thought of either in terms of *implication* or *specialisation*. Under the first interpretation, the fact that a given "real-world" entity may be appropriately described by the antecedent description is taken to *logically imply* that the entity in question may also be described by the consequent. From the (formally equivalent) alternative viewpoint, the antecedent is considered to be a *special case* of the consequent.

Our system places no restrictions on the form of the antecedent and consequent descriptions. In particular, it is thus possible to formulate AC pairs which express *simple abstractive relationships* (the fact that a *Tape drive* is a special kind of *Storage device*, for example), and *abstractive expansions* (e.g., that a *Tape drive* is a *Storage device* whose *medium* is *magnetic tape*), along with a number of more general sorts of structural and abstractive relationships. It should be noted that the relationship expressed in AC pairs is *transitive*. As it happens, the transitivity of implication is one of the characteristics which impose certain special requirements on the matching procedures. In section 2.4, we will see how this sort of requirement is accomodated by the LSEC algorithm.

2.4 Logic, Relations and Retrieval

As noted in Section 1.4, predicate logic and the relational algebra together provide a critical link between our knowledge-based retrieval language and the underlying database machine. In this section, the connections between these two mathematical tools, and their use in both conventional database management and knowledge-based retrieval tasks, will be introduced. Subsection 2.4.1 reviews those aspects of the relational model of data that are essential to our dissertation research. In Subsection 2.4.2, we illustrate the simple procedure by which pattern and target descriptions, along with all antecedent-consequent pairs to be included in the knowledge base, are *normalised*—that is, converted into relational form for manipulation using the logical and relational algebraic tools. The relationship betwen predicate logic and the relational algebra is introduced in Subsection 2.4.3, using a simple example involving the evaluation of an ordinary (as opposed to knowledge based) database query for purposes of illustration. The full set of relational algebraic primitives of concern to our thesis research are described more systematically in Subsection 2.4.4. The remaining three subsections describe the tools that are used to define the rules for matching knowledge-based descriptions, ending with a discussion of the 26 matching axioms. These axioms comprise the match specification actually used in our working demonstration system; they are included as an Appendix of this dissertation.

2.4.1 The relational model of data

The relational model of data, as typically formulated by researchers in database management systems, has its roots in two seminal papers by Codd [1970, 1972]. In this context, the term *relation* is used to denote a set of structured entities called *tuples* which, within a single relation, share a common *attribute structure*. More formally, we may define a *normalised relation* of degree n as a set of *tuples*, where each tuple is an element of the cartesian product of n (not necessarily distinct) sets--called the *underlying domains* of the relation—of non-decomposable entities. Since relations are sets, we may refer to the number of elements—in this case, tuples--in a relation as the *cardinality* of that relation. Intuitively, relations may be thought of as "tables", in which each "row" represents one tuple and each column represents one of the n *(simple) attributes* of that relation.

It is conventional to either name or number the attributes of a relation for convenience in referring either to a whole "column" of the relation, or to the *value* of the attribute in question within a particular tuple. (In some practical database applications, it may be useful to allow the appearance of "null values" in various attributes of certain tuples in the case where certain information is not available; apart from brief mention of one complication introduced by this convention, however, we will not be concerned in this dissertation with the problems of null values.) In some discussions (and in particular, in much of this thesis), it is also useful to group several attributes (some possibly repeated) together, referring to them jointly as a *compound attribute*.

The term *normalised* reflects the "type distinction" between underlying domain elements, which may serve as the values of attributes, and tuples and relations, which may not. A single tuple thus can not be used to directly rep-

resent a hierarchically nested data structure. (Our use of the term *normalised* corresponds to what is now commonly referred to as *first normal form* in the database management community. Since the introduction of the first normal form, *second* and *third* normal forms have also been proposed, each imposing successively more stringent restrictions on a set of first normal form relations in an attempt to eliminate certain kinds of anomalies which might otherwise be introduced in the course of modifying the database. Such "higher order" normal forms, however, are not relevant to the concerns of this dissertation.)

The example binary (order two) relation shown below expresses a part-whole relationship between airplanes and their (hypothetical) constituent parts:

PRODUCT	*PART*
DC-10	*wheel*
DC-10	*engine-mount*
DC-3	*oxygen-mask*
DC-10	*oxygen-mask*
DC-10	*radio*

This example relation, which will be used several times in the remainder of this section, should be interpreted as asserting that a DC-10 includes as parts a wheel, an engine mount, an oxygen mask and a radio, while a DC-3 includes an oxygen mask. Note that each of the entries in the table are "primitive domain elements"; rather than represent the fact that the DC-10 contains each of these four parts by associating an explicit *list* of parts with a single entry for DC-10, the airplane's identity must be repeated for each part in order to satisfy the normalization requirements. As we shall see in the next subsection, however, data structures expressed as lists, trees, etc. may be easily "normalized" in a simple and mechanical fashion.

2.4.2 Normalisation of knowledge-based descriptions

Upon casual inspection, our knowledge-based descriptions already appear to have the basic structure of relations. To be sure, the attribute/value structure observed in the tuples of a relation are close analogues, both in form and function, of the slot/filler constructs of our knowledge-based description language. Note, however, that while the value of a simple attribute within a given tuple must be a non-decomposable element of a particular primitive domain, the filler of a slot may itself be a general description, possibly containing its own embedded subdescriptions, and so on. Furthermore, the inclusion within a slot of descriptors of any type but individual, along with the capability for multiple abstraction through the use of multiple-descriptor descriptions, also violates the constraints of relational normalization.

As noted in Section 2.2.3, the capacity for multiple abstraction, and for an arbitrary degree of structural embedding, is essential to the kind of knowledge-based retrieval with which we are concerned. This difference between the schematic structures embodied in our knowledge-based descriptions, on the one hand, and the tuple of a normalised relation, on the other, is thus quite fundamental to our approach to knowledge-based retrieval. Consequently, although it is certainly tempting to apply the tools of logic and relational algebra more directly to the task at hand, all knowledge-based descriptions must be converted to normalised relational form before these tools can be applied. We will thus now examine the procedure by which our demonstration system normalises the knowledge-based descriptions prior to the application of logical and relational algebraic mechanisms.

Normalisation of the *external* (unnormalized) form of a description involves the addition of new tuples to various relations in the single exten-

sional database which embodies all the descriptive information manipulated by the system. (The term "extensional" will be motivated in Subsection 2.4.5.) Consider, for example, the following description, which is composed of a single perspective descriptor having *Storage-device* as its prototype and *Mag-tape* as the filler of its *medium* slot:

a *Storage-device*
  medium = *Mag-tape*

The *internal* (normalized) form of this description comprises the following set of six tuples, which would be added to five distinct relations in the extensional database (since the relation ◂DTOR--DTYPE▸ acquires two new tuples):

◂*DTION--DTOR*▸ *(G001, G002)*
◂*DTOR--DTYPE*▸ *(G002, Perspective)*
◂*PERSPECTIVE--PROTOTYPE*▸ *(G002, Storage-device)*
◂*OBJECT--SLOT--FILLER*▸ *(G002, Medium, Storage-device)*
◂*DTOR--DTYPE*▸ *(G003, Individual)*
◂*INDIVIDUAL-DTOR--INDIVIDUAL*▸ *(G003, Mag-tape)*

(In the interest of brevity, the word *DESCRIPTION* has been abbreviated as *DTION*, *DESCRIPTOR* as *DTOR* and *IMPLIES* as *IMP*, as in our actual

system. We will also be using the abbreviations *PAT*, for *PATTERN*, and *TAR*, for *TARGET*.)

Note that, in order to explicitly indicate to which relation the tuples in question will be added, we have used a somewhat different representation scheme, called *intensional* notation, for representing the normalised form of our example description. In intensional notation, the name of the relation appears first, followed by a parenthesized list which specifies (in a predetermined order associated with the relation in question) the value of each simple attribute in the particular tuple being represented. Particular tuples in intensional notation thus have the form of logical *predicates* (with constant arguments), which, as we shall see in the following subsection, will permit their incorporation in well-formed formulæ of a restricted first order predicate calculus. The significance of the asterisks surrounding the relation name in intensional notation will be discussed in Subsection 2.4.5.

While it will not be necessary to consider the detailed semantics of each of the sixteen relations which are employed in our demonstration system, and their relationship to the external descriptions from which they are generated, it is worth noting a few characteristic differences between the external and internal forms of a description. Perhaps most obvious is the appearance within the internal form of a number of primitive elements (those of the form *G00n* in our example) which may be said to "carry meaning" solely on the basis of their relationship with other, explicitly named, primitive elements. (This technique, along with the naming convention for such elements, should be familiar to LISP programmers as an analogue of the "GENSYM-ed" atom.)

More interesting, however, is the appearance within the internal form of certain "naturally" named primitive elements which nonetheless do not appear explicitly within the original external description. In particular, note

that while some of the named elements correspond to *semantically* meaningful tokens found in the external description, others (in our example, *Perspective* and *Individual*) serve a purely *syntactic* function, explicitly representing in normalised relational form the structural information that was implicit in the syntax of our knowledge-based description language. Loosely speaking, the process of normalization involves a *flattening* of the original tree structure of the external description, together with an *expansion* of the original description to explicitly represent syntactic information.

Since descriptions serve roles as patterns, targets, antecedents or consequents within our system, tuples must be added to the extensional database to reflect these roles. If our example were in fact a target description (which of course seems unnatural in this case), this connection would be drawn by the addition of the single tuple

•TARGET--COLLECTION• (G001, (collection name))

to the *•TARGET--COLLECTION•* relation. If, on the other hand, the example description were the consequent of an AC-pair whose antecedent description was a *Tape-drive*, the following four tuples would instead be added to the extensional database:

•ANTECEDENT--CONSEQUENT• (G004, G001)
•DTION--DTOR• (G004, G005)
•DTOR--DTYPE• (G005, Perspective)
•PERSPECTIVE--PROTOTYPE• (G005, Tape-drive)

Note that the knowledge base, which was characterized in Chapter 1 as a separate database for pedagogical reasons, is actually implemented as part of the single extensional database, together with all candidate target descriptions in the collection and the internal form of the pattern description used in the course of a particular retrieval session.

2.4.3 Logic as an effective query language

In this subsection, we will examine some fundamental connections between the descriptive capabilities of the first order predicate calculus and the use of relational algebraic operators to make such logical descriptions computationally effective. While we might have chosen to exemplify this relationship using a realistic example of the behavior of our demonstration system, the relative complexity of the knowledge-based matching operation would have made it unnecessarily difficult to identify the essential import of such an example. We have thus chosen to illustrate the central aspects of our use of mathematical logic and relational algebra through the use of a simple example typical of conventional (as opposed to knowledge-based) retrieval in a relational database management context. Of interest in this regard is the observation that a wide range of typical database queries can be expressed in the form of

1. a well-formed formula in the first-order predicate calculus (without functions), together with

2. a list of free variables of that well-formed formula, all possible joint instantiations of which are to be returned as the value of the query.

This observation is best illustrated by considering a simple example. To this end, we have supplemented the hypothetical *•PRODUCT--PART•* relation introduced in Subsection 2.4.1 with a new *•CUSTOMER--PRODUCT•* relation, which is to be interpreted as asserting that American Airlines has purchased one or more DC-3's and one or more DC-10's, while Western Airlines owns one or more DC-10's only:

CUSTOMER	*PRODUCT*
American	*DC-3*
Western	*DC-10*
American	*DC-10*

PRODUCT	*PART*
DC-10	*wheel*
DC-10	*engine-mount*
DC-3	*oxygen-mask*
DC-10	*oxygen-mask*
DC-10	*radio*

Suppose we wished to produce a list which paired each airline with each part which could be found in the inventory of that airline, independent of the identity of the model (or models) of airplane which accounted for the presence of that part within the inventory of that airline. In relational terms, we should like the result of our query to be a new binary relation having two attributes—one ranging over the same primitive domain as the *CUSTOMER* attribute of the *•CUSTOMER--PRODUCT•* relation, and one over that of the *PART* attribute of the *•PRODUCT--PART•* relation—each of whose tuples satisfies the relationship in question.

Such a query might be expressed in the following way using the language of first-order logic:

$$(x, z): \exists y\ (\bullet CUSTOMER\text{--}PRODUCT\bullet\ (x, y) \wedge \bullet PRODUCT\text{--}PART\bullet\ (y, z))$$

where the *result variables* are specified in a parenthesized list which is prefixed to the well-formed formula and followed by a colon. Here, we are assigning a correspondence between the *predicate* *•CUSTOMER--PRODUCT•* and the

relation having *CUSTOMER* and *PRODUCT* as its attributes, and similarly, between the *•PRODUCT--PART•* predicate and the other relation. The *result* of this query is defined to be a new relation, having two attributes (corresponding to the two free variables x and z specified in the result variable list), whose tuples enumerate all of the combinations of one instantiation of x and one instantiation of z for which the well-formed formula is true for some instantiation of the existentially quantified variable y.

Let us now consider how the result of such a query might be computed. All possible combinations of tuples, one of which is chosen from the *•CUSTOMER--PRODUCT•* relation, the other from *•PRODUCT--PART•*, whose *product* attributes share a common value are identified, as illustrated below by the lines connecting tuples of the two argument relations:

CUSTOMER	PRODUCT
American	DC-3
Western	DC-10
American	DC-10

PRODUCT	PART
DC-10	wheel
DC-10	engine-mount
DC-3	oxygen-mask
DC-10	oxygen-mask
DC-10	radio

For each such matching pair of tuples, a new tuple is created by concatenating the two and eliminating one copy of the common *PRODUCT* attribute, thus yielding the following ternary relation:

CUSTOMER	PRODUCT	PART
American	DC-3	oxygen-mask
Western	DC-10	wheel
Western	DC-10	engine-mount
Western	DC-10	oxygen-mask
Western	DC-10	radio
American	DC-10	wheel
American	DC-10	engine-mount
American	DC-10	oxygen-mask
American	DC-10	radio

The operation that we have just described provides our first example of a relational algebraic operation, which is called the *join* (more precisely, the *natural join*) of the two argument relations. The *PRODUCT* attributes of each of the two argument relations are together referred to as the *join attributes*.

Recall, however, that our formulation of the query made no reference to the product by which the customer and part are related. To produce the desired result relation, we must therefore remove the *PRODUCT* attribute from our intermediate result. Notice, however, that the first and eighth tuples in the intermediate result are distinguished only by the value of their respective *product* attributes. Upon removal of this attribute, these two tuples would no longer differ, introducing a redundancy in the result relation which is prohibited by the fact that relations are sets. (As we shall see in the following subsection, our injunction against relations with redundant tuples does not reflect a superstitious adherence to our formal definition of relations, but is in fact motivated by important practical considerations.)

The final step of our example thus involves not only removal of the *PRODUCT* attribute, but also elimination of the redundant tuples that would otherwise result from the removal of formerly distinguishing attribute values:

CUSTOMER	*PART*
American	*oxygen-mask*
Western	*wheel*
Western	*engine-mount*
Western	*oxygen-mask*
Western	*radio*
American	*wheel*
American	*engine-mount*
American	*radio*

This combined operation is an example of another relational algebraic operation, called *projection*. We refer to the *CUSTOMER* and *PART* attributes, which are carried through to the result relation in our example, as *projected attributes*, while the *PRODUCT* attribute is described as *projected out* of the argument relation.

The very simple example that we have just considered has required only the informal introduction of the join and project operators. In the following subsection, we will define a larger set of relational algebraic primitives, providing a more rigorous definition for each one. The importance of the relational algebra to our task (and indeed, to the "higher level" goals outlined in Chapter 1), derives from the fact that this more complete set of relational algebraic primitives has all the "descriptive power" of the logic-based query language introduced above.

Perhaps the best known analogue of this observation in the relational database literature is the so-called *completeness* result due to Codd [1972]. In essence, Codd gave a constructive proof that any query formulated using the *calculus of n-expressions* (often called the *relational calculus*)—a descriptive language quite similar to our own first-order predicate calculus-based query

language, but where, among other distinctions, quantification is over tuples and not elements of the primitive domains—could be computed by application of a suitable sequence of relational algebraic primitives. Ignoring for the moment the differences between strict first-order logic and relational calculus, Codd's result thus provides a systematic (though generally inefficient) way of computing the result of an arbitrary first-order query, as defined above, using only the relational algebraic operations that will be defined in the next subsection.

One way of viewing the roles of logic and relational algebra in this sort of retrieval task that we have found particularly useful in our work is based on the notion of a *formal theory*. Within this theoretical framework, we view the query as part of a *first-order theory*, and the relations in the (extensional) database as a particular *finite interpretation* of that theory. In particular, each primitive predicate in the query is associated with one relation in the extensional database, which is treated as its *interpretation*. Within this framework, we can view the problem of finding the result of the query as that of finding all possible values of the (free) result variables such that the query well-formed formula is *logically satisfied* under that interpretation. For this reason, we sometimes call refer to the task of computing the result of a logical query as one of *satisfaction* over a finite (albeit generally large) domain.

2.4.4 The relational algebraic primitives

The relational algebra we have used in our dissertation research is based on a small set of algebraic operators enumerated by Codd [1972] which take one or more relations (along with certain "control" information) as arguments, returning a single new relation as their value. This set of primitives includes the ordinary set operations—which, with one restriction, are defined for relations in much the same way as for other sets—along with several *structured* operators, which make reference to the internal attribute structure of the constituent tuples. The two most important structured operators, *project* and *join*, have already been informally described in the previous subsection. Several other structured operations will also be introduced in this subsection which may in fact be derived from project, join and the unstructured set operations, but which serve certain particularly important functions in many practical applications of relational algebraic systems.

Specifically, we will be concerned in this dissertation with the following relational algebraic primitives:

1. Union
2. Intersection
3. Set difference
4. Projection
5. Join
7. Selection
8. Restriction

The three binary set operators *union*, *intersection* and *set difference* are defined in a relational algebraic system in the same way as for sets in general, with one exception: the relational version of each is defined only when the two

relations that serve as its operands are *union-compatible*. Two relations are said to be union-compatible if and only if they are of the same degree n and the underlying domains of the i-th simple attributes of the two relations are the same for all i, $(1 \leq i \leq n)$.

We thus define the union of two union-compatible relations R_1 and R_2, denoted $(R_1 \cup R_2)$, as a relation consisting of exactly those tuples that are an element of R_1, of R_2, or both. The intersection $(R_1 \cap R_2)$ is defined as that relation containing all tuples found in *both* R_1 and R_2. Finally, the set difference $(R_1 - R_2)$ is defined to consist of exactly those tuples of R_1 that are *not* present in R_2.

In preparation for our formal definition of the projection operator, we will need to introduce some additional notation. First, we adopt the convention that a list of primitive domain elements enclosed by angle brackets ("⟨" and "⟩") will designate a new tuple containing the specified elements as the values of its simple attributes, in the order listed. Furthermore, if r is a tuple of some n-ary relation R, we will define $r[j]$ to be the value of the j-th attribute of r, $(1 \leq j \leq n)$. It will be convenient to extend this notation to allow expressions such as $r[A]$, where A is a *compound* attribute of R consisting of the m (not necessarily distinct) simple attributes numbered $j_1, j_2, \ldots, j_m$, defined such that $(r[A])$ represents the new tuple $\langle r[j_1], r[j_2], \ldots, r[j_m] \rangle$.

We may now define the projection of a relation R over the compound attribute A as the set

$$\{(r[A]) : r \in R\}$$

Note that we have defined the projection operator in such a way that simple attributes within the compound attribute A may be *replicated* in the course of projection. Depending on certain details in the definition of the join operation,

this convention may have important theoretical consequences affecting the expressive power of the resulting algebra

The projection operator may be thought of as a sort of "vertical subsetting" operation, in which

1. the "non-projected" attributes of each tuple in the argument relation are eliminated,
2. the remaining attributes may be permuted and/or replicated, and
3. any duplicate tuples that result from the elimination of values that formerly distinguished different tuples are then removed.

In most implementations on a von Neumann machine—that is, a "conventional" computer system having a single central processing unit acting on a single bank of random access memory—the attribute elimination and permutation/replication functions can both be implemented using a simple and computationally inexpensive procedure whose complexity is linear in the cardinality of the argument relation. The elimination of redundant tuples, on the other hand, may be surprisingly time-consuming, particularly when the argument relation is large. In fact, one common convention in some von Neumann implementations is to relax the requirement that relations be true sets, allowing the introduction of duplication during some or all projections. This approach introduces the following problems, however:

1. the maintenance of duplicate tuples may lead to combinatorially explosive growth in the cardinality of the intermediate results of a complex query, and
2. functions sensitive to the repetition of identical tuples—the calculation of numerical counts and statistical measures, for example—will not yield accurate results if redundant tuples are not first eliminated.

One of the goals of the architectures discussed in Section 3 is the implementation of true projection without the high cost of redundant tuple

elimination.

Definition of the join operation requires the definition of one additional construct: the *concatenation* of two tuples. If r_1 is a tuple of a relation R_1, having degree n_1, and r_2 is a tuple of relation R_2, having degree n_2, the *concatenation* $(r_1|r_2)$ of r_1 and r_2 is defined to be the new $(n_1 + n_2)$-tuple

$$(r_1[1], r_1[2], \ldots, r_1[n_1], r_2[1], r_2[2], \ldots, r_2[n_2]) \quad .$$

Several variations of the join operator are commonly discussed in the literature; we will begin by defining a particularly important variant known as the *equi-join*. The equi-join of two relations R_1 and R_2 over the compound attributes A_1 and A_2, respectively (each assumed to be composed of the same number of simple attributes, with corresponding simple attributes having underlying domains that are comparable under the equality predicate) is defined as

$$\{(r_1|r_2) : r_1 \in R_1 \wedge r_2 \in R_2 \wedge r_1[A_1] = r_2[A_2]\} \quad .$$

A_1 and A_2 are referred to as the (compound) *join attributes*, which will have special significance in the architectures introduced in this chapter. In the case where A_1 and A_2 are the degenerate compound attributes containing no simple attributes, equi-join reduces to the *extended cartesian product* of the tuples of R_1 and R_2—that is, to the set of all possible concatenations of one tuple from R_1 with one from R_2. The more general join operation may be intuitively thought of as a process of filtering the extended cartesian product of R_1 and R_2 by removing from the result all conjoined tuples whose respective join attributes have different values. (The computational method suggested by this interpretation, of course, would in general be impractically inefficient.)

It should be acknowledged that our definition of the equi-join operator leaves unanswered certain questions which, although not immediately relevant

to the concerns of our research, are commonly encountered in practical database applications. One such problem arises in the case where null values are allowed to appear in the join attributes, since it is generally not appropriate to treat two tuples as matching in the case where their join attributes are both null. Consideration of the various approaches that have been advanced for the accomodation of this case will not fall within the scope of this dissertation, however.

The join operation is in general extremely expensive on a conventional von Neumann machine, since the tuples of R_1 and R_2 must be paired for equality with respect to the join attributes before the extended cartesian product of each group of "matching" tuples can be computed. In the absence of physical clustering with respect to the join attributes (whose identity may vary in different joins over the same pair of relations), or the use of various techniques requiring a large amount of redundant storage, joining is typically accomplished most efficiently on a von Neumann machine by pre-sorting the two argument relations with respect to the join fields. The *order* of the tuples following the sort is actually gratuitous information from the viewpoint of the join operation. From a strictly formal perspective, the requirements of a join—that the tuples be paired in such a way that the values of the join attribute *match*—are significantly weaker than those of a sort, which require that the resulting set be *sequenced* according to those values. The distinction is moot in the case of a von Neumann machine, where no better general solution to this **pairing problem** than sorting is presently known. One of the design goals of the architecture described in Chapter 3, however, is to make use of the weaker constraints involved in the definition of the join operation to obviate the need for either pre-sorting or the extravagant use of redundant storage.

One common variant of the equi-join operator is the *natural join*, intro-

duced in the example of the previous section, in which one of the two join attributes, which are redundantly represented in the result relation in the case of equi-join, is eliminated (as if by projection). Our architecture supports the natural join with a simple modification of the internal equi-join algorithm which will be described in subsection 3.4.2. A more general form of join often discussed in the literature is the θ-join, whose definition is similar to that of the equi-join, but with the equality predicate replaced by a more general binary predicate θ. (In Codd's definition, θ is defined to be one of the arithmetic operations $=$, $\neq$, $<$, $\leq$, $>$ or $\geq$.) Considerations for the efficient evaluation of the general θ-join operator differ in several respects from those involved in evaluating the equi-join. We will not discuss this more general case in the present dissertation.

Each of the other relational algebraic operators which will be described in this subsection can in fact be derived from the structured operators project and join and the three unstructured set operators, and are defined here for one or both of the following reasons:

1. The operator embodies a special case of one or more of the previously defined primitives which might admit the possibility of either a less complex, or a more efficient, hardware implementation.
2. The operator represents an important and frequently encountered use of some composition of the primitives defined earlier.

One derived operation that occurs frequently in both practical and theoretical discussions, and which has a special role in most of the hardware designs that we will discuss, is called *selection*. Most algorithms and architectures for "associative retrieval" implement what is essentially a process of relational selection. The select operator also plays an important role in the architectures we propose in Chapter 3, although unlike most associative processor

designs, our architecture explicitly addresses the problems of efficiently implementing other relational operators as well. The select operator returns a subset of its single argument relation consisting of all tuples that satisfy a list of attribute/value pairs. The *select* operator may thus be regarded as a natural join of the argument relation with a *singleton* relation (a relation consisting of exactly one explicitly specified tuple) over all attributes of the singleton. More precisely, the result of a selection from relation R with compound attribute A and value tuple V is

$$\{r : r \in R \wedge r[A] = V\} \quad ,$$

where the corresponding A and V domains are again assumed to be compatible with respect to equality.

Another important derived operation is known as *restriction*. While restriction, like the join operator, is sometimes defined in terms of a general θ, we will again be concerned only with the case where θ is the binary equality predicate. The restriction of a relation R over the compound attributes A_1 and A_2 (both composed of simple attributes of R) is defined as

$$\{r : r \in R \wedge r[A_1] = r[A_2]\} \quad .$$

In its most common form, where the compound attributes A_1 and A_2 are each composed of exactly one simple attribute, the restriction operator returns all tuples of its argument relation in which the values of the two specified attributes are equal. Although restriction can be defined solely in terms of the join and project operators, an implementation based in a straightforward way on this derivation would be considerably more complex and inefficient than one specifically tailored to support the restrict operator. Restriction is an important enough operation in practice that we have treated the capacity for

direct (and efficient) evaluation of restrictive expressions as a significant design objective.

Finally, we must acknowledge a derived operation that has considerable theoretical and practical importance in many applications, but to which we have devoted little special attention in our evaluation of alternative architectures. This operation, called *division*, is used to achieve the effects of universal quantification within the queries of a language based on the relational calculus (Codd [1972]) and may well be worthy of special attention in course of designing a generally-applicable relational database machine. Since it was not necessary that this kind of operation be implemented efficiently in its full generality for purposes of our thesis application, however, the relational division operator will not be given the same sort of special consideration in this dissertation as the other two derived operators described above.

2.4.5 Defined predicates

The example query formulated in Subsection 2.4.3 made use of two logical predicates, each associated with an explicitly defined relation that might be stored in an "extensional database" of the sort used in our demonstration system. We refer to a predicate of this kind, whose meaning derives from its association with a relation whose constituent tuples are explicitly enumerated, as a *primitive predicate*. As we shall see in Subsection 2.4.7, though, our knowledge-based retrieval task requires the use of another kind of predicate, which we will call a *defined predicate*.

A defined predicate corresponds to no fixed, explicitly defined relation, as does a primitive predicate, but instead derives its meaning from a *proper* (non-logical) *axiom* expressing its equivalence to a well-formed formula involving other (defined or primitive) predicates. (Our notion of a defined predicate is closely related to that of a *view*, as defined in the relational database literature, and to other constructs that have been introduced periodically by researchers in other areas of computer science; it is our use of defined predicates that should be of interest here.) Following a convention employed in certain recent work on logic and databases, we will sometimes refer to the set of defined predicates as the *intensional database* of our system (by contrast with the extensional database, which is composed of relations specified explicitly by enumerating all their tuples, each of which corresponds to a primitive predicate in the logical query language). In the interest of perspicacity, we adopt the convention of surrounding the name of each primitive predicate by asterisks, as in the examples we have already seen, while the names of defined predicates will not be so enclosed.

By way of illustration, let us consider a very simple example of a defined

predicate, called *CUSTOMER--PART*, whose body is precisely the well-formed formula from the sample query introduced in Subsection 2.4.3:

$$CUSTOMER\text{--}PART(x, z) \equiv$$
$$\exists y.$$
$$(*CUSTOMER\text{--}PRODUCT*(x, y) \wedge$$
$$*PRODUCT\text{--}PART*(y, z))$$

In this example, the defined predicate *CUSTOMER--PART* is defined in terms of the two primitive predicates $*CUSTOMER\text{--}PRODUCT*$ and $*PRODUCT\text{--}PART*$, allowing its *reduction* to predicates whose interpretations are explicitly available in the extensional database. The newly defined predicate could thus be used as a sort of "shorthand" for the sample query, which might now be expressed as

$$(x, z):$$
$$CUSTOMER\text{-}PART(x, z) \quad .$$

Although intensionally defined predicates (more precisely, their analogues within the calculus of α-expressions) were not included as part of Codd's original formalism for the relational model of data, more recent work by several researchers suggests one possible manner in which defined predicates might be employed within a first-order query. At the risk of oversimplification, this

approach (exemplified by Reiter [1977] and by Chang and Lee [1973]) involves a two step procedure for the evaluation of queries. During the first step, the query (which may involve both primitive and defined predicates), along with the predicate definition axioms found in the intensional database, is manipulated using automatic theorem-proving techniques to remove all occurrences of the defined predicates, yielding a transformed query containing only primitive predicates. The resulting transformed query is then evaluated using ordinary database retrieval techniques. As we shall see in the next subsection, however, the lack of extensional feedback in the course of intensional manipulation introduces certain problems which preclude the possibility of a straightforward application of this approach to our thesis task.

2.4.6 Recursive predicate definition

From the perspective of our own application, one of the most serious limitations of the approach to retrieval that we informally outlined at the end of the last subsection relates to the need to support *recursively*-defined predicates. Before considering the role of recursive predicate definition within our system for knowledge based retrieval, let us consider a very simple example of a recursively defined predicate. Imagine for the moment that the extensional database contained a binary relation, called *'CHILD--PARENT'*, asserting that *Suzanne* has *Charles-Jr* and *Marilyn* as parents, while *Charles-Jr* is in turn the son of *Charles-Sr* and *Estelle*, etc., as illustrated below:

CHILD	*PARENT*
Suzanne	*Charles-Jr*
Suzanne	*Marilyn*
Charles-Jr	*Charles-Sr*
Charles-Jr	*Estelle*
Marilyn	*Benjamin*
Marilyn	*Esther*

Suppose now that we wished to construct a defined predicate *DESCENDANT--ANCESTOR* having the value true whenever its first argument is either the child, grandchild, great-grandchild, etc. of its second argument.

If it were not for the problem of recursion within a predicate definition, the *DESCENDANT--ANCESTOR* predicate could be defined as

$$DESCENDANT\text{--}ANCESTOR(x, z) \equiv$$
$$\quad \bullet CHILD\text{--}PARENT\bullet\ (x, z) \vee$$
$$\quad \exists y.$$
$$\quad\quad (\bullet CHILD\text{--}PARENT\bullet\ (x, y) \wedge$$
$$\quad\quad\ \bullet DESCENDANT\text{--}ANCESTOR\bullet\ (y, z))$$

Note, however, that the first step (query translormation) of the hypothetical two-step process outlined in the previous section could no longer simply replace each occurrence of the defined predicate *DESCENDANT--ANCESTOR* with its body, since that body itself contains another invocation of *DESCENDANT--ANCESTOR*, the theorem-proving technique would thus either fail to terminate, or fail to find all possible results of the query.

The approach that we adopted in the LSEC algorithm to the problem of recursive predicate definition uses intermediate results of the query evaluation in order to terminate computation after all potentially relevant results have been obtained, avoiding computational loops based on the endless expansion of cycles of mutually-defined predicates. While we will not examine the details of the LSEC algorithm at this point, the basic mechanism by which LSEC handles recursion is quite simple. In essence, our approach involves the treatment of the conjunctive operator in a query well-formed formula as a "computational" (as opposed to a strictly "logical") *AND*. As in the case of the LISP *AND*, the operands of such a conjunction are evaluated in left-to-right order until the first "failure" - specifically, until the first case in which application of the LSEC procedure to some operand yields a *false-extension* (defined rigorously in Section 2.5), which may regarded as a generalization of the boolean constant

false. In the above example, •DESCENDANT--ANCESTOR• (y, z)) will not be recursively evaluated after the primitive predicate (•CHILD--PARENT• (x, y) is found to have no possible instantiations consistent with the current binding of x—that is, after reaching the level of the "grandparents".

In passing, it should be acknowledged that the importance of recursive predicate definition within our own application may well reflect characteristics specific to our thesis task. We are thus unable to make any claims regarding the suitability of our approach for use in conventional relational database applications. Indeed, Minker [1978], after reviewing the problems of recursive predicate definition, suggests that "it is not clear that one should permit recursive axioms (in the intensional database) for realistic problems". The suspicion that the additional theoretical power provided by recursive predicate definition may have minimal significance in the context of contemporary practical database requirements has been echoed by other authors. While our own areas of expertise do not permit a qualified opinion in this regard, it should be emphasized that in *our* application, in which defined predicates are not directly incorporated in a user-level description language, but instead are used internally to express and implement the semantics of a higher-level description language, *recursive predicate definition is essential*.

In the following subsection, we will examine the way in which recursively defined predicates in the intensional database are used to define the semantics of our knowledge-based description language.

2.4.7 Axioms defining the match semantics

Having now introduced the essential logical and relational tools, let us now consider the manner in which a user's knowledge-based pattern description is matched against the collection of candidate target descriptions according to the axioms defined in the match specification. In each such query task, regardless of the particular pattern description, the "top-level" logical query is has the form

(x): *DTION-IMP-DTION(x, pat-dtion)* .

The intended result of this query is a unary relation, each of whose tuples has as the value of its single attribute a particular target description (more precisely, the primitive domain element that anchors the internal form of that target description) which matches the given pattern description according to the rules defined in the match specification. Specifically, the match specification comprises a definition of the defined predicate *DTION-IMP-DTION* in terms of other predicates, some of which are themselves defined in other axioms, and so on.

The names of each of the 26 axioms defining the match semantics of our knowledge-based description language are listed in Figure 2.2. We have indented this list to indicate which predicates are defined in terms of which other predicates, with the name of a defined predicate shown below, and indented by one unit with respect to, the name of the defined predicate in whose body it is first used. (The "top-level" defined predicate *DTION-IMP-DTION*, for example, is defined in terms of the defined predicate *DTION-DIRECTLY-IMP-*

DTION.) Exceptional definitional directions are indicated explicitly using arrows (*PERSPECTIVE-DTOR-IMP-PERSPECTIVE-DTOR*, for example, is defined in terms of the top-level defined predicate *DTION-IMP-DTION*.) Each of the upward pointing arrows thus identifies a recursive predicate definition loop, underscoring the central role such definitions occupy in our thesis system.

Although it will not be necessary at this point to consider the details of each defined predicate in the match specification, it may be instructive to consider one typical such predicate in an attempt to convey some feeling for the kind of information embodied in these matching rules. To this end, we consider the defined predicate *PERSPECTIVE-DTOR-IMP-PERSPECTIVE-DTOR*, which implements what we may call *syntactic perspective matching*. In the course of syntactic perspective matching, all target perspectives that match a given pattern perspective and have the same prototype as that pattern perspective are identified. By contrast with *semantic perspective matching*, this procedure does not identify those target perspectives that have *different* prototypes than that of the pattern, but that would in fact satisfy the matching criteria on the basis of domain-specific specialization relationships derived from the knowledge base. The meaning of this defined predicate is reasonably straightforward: in order for a "target-perspective-dtor" to match a "pattern-perspective-dtor" on the basis of this defined predicate, the prototypes of each must be the same, and for each slot which is filled in the pattern, the corresponding slot must be filled in a compatible way (as specified by the recursive call to *DTION-IMP-DTION*) within the target.

The full set of 26 axioms that together consitute the match specification for our knowledge-based description language are presented in the Appendix.

Having now examined an *example* of the use of well-formed formulæ within our demonstration system, it seems appropriate to explicitly list the

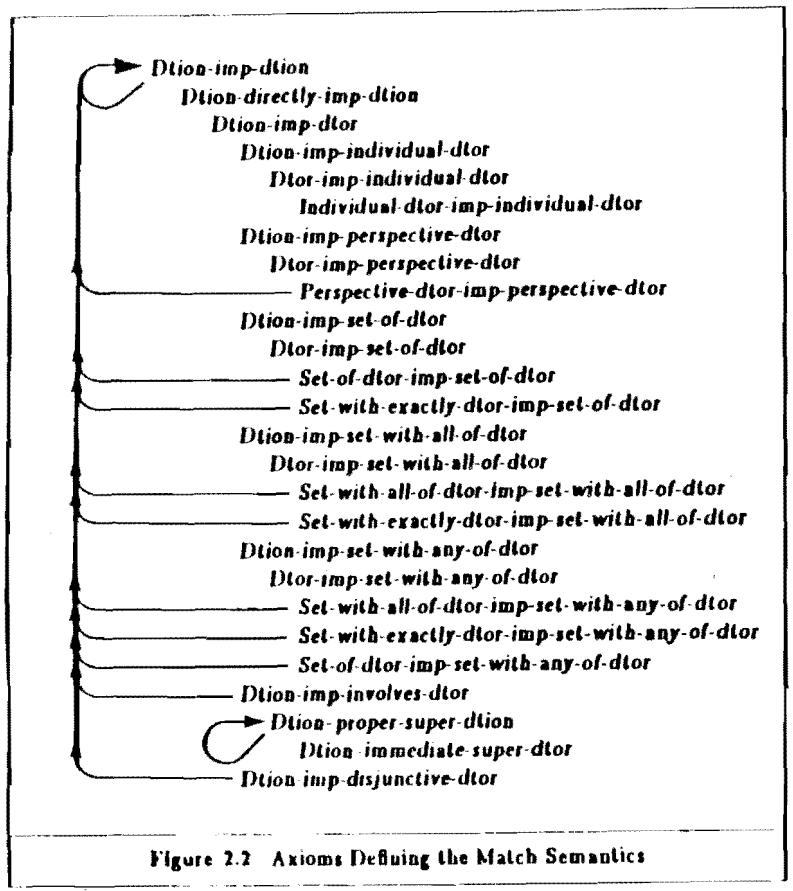

Figure 2.2 Axioms Defining the Match Semantics

respects in which our logic-based query and predicate definition language is in fact restricted by comparison with the full language of first-order predicate calculus. First, function symbols (which in fact add no formal expressive power to the predicate calculus) have been eliminated from consideration. Second, our present application has not required the use of explicit *negation*, the *NOT* operator has thus been omitted from our language as well. Note

that by contrast with the case of functions, the addition of negation would significantly expand the expressive capabilities of our language; we consider such an extension to represent a potentially important avenue along which our own work might be extended. Because negation introduces a number of surprisingly difficult problems when used in a computationally effective description language, however, we have chosen to omit this construct from consideration in our thesis work.

Finally, we have intentionally permitted only a restricted form of universal quantification. Rather than permit universal quantification of the general form

$$\forall x.P(x) \quad ,$$

we restrict $P(x)$ to be of the form

$$Q(x) \supset B(x) \quad ,$$

where $Q(x)$, called the *qualification clause*, is further restricted to contain no disjunction or universal quantification, while the *body* $B(x)$ is an unrestricted well-formed formula. While detailed consideration of universal quantification will be deferred to Section 2.5, we note for now that the qualification clause serves the important practical function of restricting the plausible range of universally quantified variables, thus limiting the set of "x values" which must be considered to those values for which $Q(x)$ is satisfied.

2.5 The LSEC Algorithm for Logical Satisfaction

In this section, we will describe the function of the LSEC algorithm, through which the connection between logical descriptive mechanisms and actual relational algebraic operations is established in our demonstration system. The LSEC algorithm has been fully implemented in our demonstration system and tested on carefully chosen examples designed to exercise each portion of the algorithm.

We begin in Subsection 2.5.1 with an example of the use of LSEC in evaluating the results of a simple conventional database query, avoiding many of the spurious complications arising in our knowledge-based retrieval application that are peripheral to the essential operation of the LSEC algorithm. In the remainder of the section, we describe the behavior of the algorithm upon encountering each of the six types of logical formulæ used in constructing the match specification, ending with the procedure by which the result relation is constructed.

2.5.1 A simple example

The example we have chosen to illustrate the process of *extensional constraint* has the virtue of illustrating the essential behavior of the LSEC algorithm, but may well seem a bit contrived to the reader. To be sure that our somewhat unnatural example does not obscure the features we will be attempting to illustrate, we thus begin with a brief discussion of its "real-world" setting.

In our example, a hypothetical capitalist wishes to affect the behavior of major U.S. corporations by exerting an indirect influence on key individuals within those corporations -- specifically, on the officers and directors of those firms. This indirect influence is in turn to be mediated by the attorneys and accountants of these key individuals, on whom these individuals are presumed to rely for information and advice. The capitalist might succeed, for example, in influencing the behavior of the corporation by bribing its president's attorney, or the accountant of one of its directors. To this end, our capitalist wishes to review a list of professionals (attorneys and accountants), paired with corporations on whom these professionals could ultimately exert an influence through some third party whose identity is of no concern to the capitalist.

Four two-attribute relations will be used in our example:

ATTY	*CLNT*
Atty1	*Jones*
Atty1	*Stone*
Atty2	*Jones*

ACCT	*CLNT*
Acct1	*Smith*
Acct1	*Jones*

OFCR	*CORP*
Smith	*XCorp*

DRTR	*CORP*
Jones	*YCorp*
Jones	*XCorp*
Smith	*XCorp*

The first associates attorneys with their clients; the second, accountants with their clients; the third, officers with the corporations by which they are employed; the fourth, directors with the corporations on whose boards they serve. The

binary relation which our hypothetical capitalist wishes to review may be described using the logical query

(*professional, corporation*): ∃ *client*.
((•*ATTY--CLNT*• (*professional, client*) ∨
•*ACCT--CLNT*• (*professional, client*)) ∧
(•*OFCR--CORP*• (*client, corporation*) ∨
•*DIRTR--CORP*• (*client, corporation*)))

Without concentrating on the details of the algorithm, we will now sketch the manner in which LSEC finds the results of this query.

The most important data structure maintained and manipulated by the LSEC algorithm is called the *Accumulated Disjoint Extension* (sometimes referred to as the *ADE*, or somewhat less precisely, as simply the *extension*. Loosely speaking, the ADE may be thought of as a set of relations, each of which describes *one way* in which that part of the logical formula thus far encountered may be satisfied. (In one sense, the ADE may thus be thought of as a sort of dynamic, extensional analogue to the intensional notion of a *disjunctive normal form*.) The initial ADE in any LSEC session is always the distinguished ADE *true-extension*, which is a set consisting of the single degenerate relation containing no attributes and no tuples (the *true-relation*). In the course of processing the top level query (and the defined predicate bodies introduced in the course of expanding that query), this ADE is *constrained* by the various logical subformulæ; after the whole query has been processed, the result relation may be extracted (in a manner detailed in Subsection 2.5.9) from the final ADE.

The tree-traversal algorithm embodied in the LSEC algorithm causes the initial ADE (*false-extension*) to be constrained first by the disjunction

(•*OFCR--CORP*• (*client, corporation*) ∨
•*DIRTR--CORP*• (*client, corporation*))

The resulting ADE is a set of two relations, the first corresponding to influences exerted through officers, and the second through directors:

Smith	*XCorp*

Jones	*YCorp*
Jones	*XCorp*
Smith	*XCorp*

Constraining this ADE further to reflect the content of the first conjoined subformula of the body of our query (to reflect the fact that access to these key individuals may be through either their attorneys or accountants), LSEC "multiplies through" the original •ATTY--CLNT• and •ACCT--CLNT• relations by *joining* each with each of the two *terms* (constituent relations) in the ADE over the common existentially quantified variable *client*. Because none of the attorneys in our example has as client the lone individual (*Smith*) who serves as a corporate officer, one of the four resulting terms will be a *false-relation* - a relation having any attribute structure, but no tuples - which will be eliminated from the ADE, yielding the three relation ADE

Atty1	Jones	YCorp
Atty1	Jones	XCorp
Atty2	Jones	YCorp
Atty2	Jones	XCorp

Acct1	Smith	XCorp

Acct1	Smith	XCorp
Acct1	Jones	YCorp
Acct1	Jones	XCorp

Notice that by existentially quantifying *client*, instead of leaving it free in the query well-formed formula, we have indicated that the identity of the client is of no concern to the end-user; it is used only to establish that *some* link between professional and corporation exists. Upon exiting from the scope of the existentially quantified formula, we thus *project* out the *CLNT* attribute from all terms in the current ADE. The result (after eliminating one redundant tuple in the third relation) is

Atty1	YCorp
Atty1	XCorp
Atty2	YCorp
Atty2	XCorp

Acct1	XCorp

Acct1	XCorp
Acct1	YCorp

Finally, the *union* of all relations in the ADE is taken, thus combining all its terms into a single result relation:

Atty1	YCorp
Atty1	XCorp
Atty2	YCorp
Atty2	XCorp
Acct1	XCorp
Acct1	YCorp

Note that during the performance of this final union operation, one more tuple becomes redundant, and must be eliminated.

By way of summary, then, the LSEC algorithm starts with an unconstrained extension (the ADE *true extension*), which is then constrained by the query formula. In our demonstration system, the initial query is always the defined predicate

(x): *DTION-IMP-DTION*(x, pat-dtion) ,

whose body is expanded in the course of the algorithm, with different types of logical formulae "surfacing" at different depths within this expansion. In the remainder of this section, we will review the behavior of the LSEC algorithm upon encountering each of these logical formula types.

2.5.2 Existential quantification

The processing of existentially quantified formulæ within the top-level query, or within some defined predicate body encountered in the course of evaluating this query, is quite simple, but provides a good introduction to the use of one of the essential data structures manipulated by the algorithm: the *logical variable stack*. (Although it will be referred to informally as simply "the stack", the logical variable stack should not be confused with the stack maintained by the underlying LISP system, which maintains the bindings of the λ-variables involved in execution of our demonstration system.)

A new stack frame is created each time an existentially or universally quantified formula, or a defined predicate, is encountered, to hold certain information about each quantified variable or formal parameter (in the case of the defined predicate) introduced within the current formula. There are two components to this information: the first specifies the variable's *type*, which may be either *constant-valued* or *attribute-id-valued*, while the second is the value itself. Quantified variables, though, are always of the latter type; upon encountering the existentially quantified formula, a unique name called an *attribute identifier* (*attribute-id*) is generated for each existentially quantified variable to designate a particular attribute that will ultimately appear in one or more relations in the ADE. Although the same attribute may be referenced using different names within the bodies of different defined predicates, each such name will be "bound" to the same attribute-id at different locations within the logical variable stack. Conversely, the same variable name may appear in various frames within the current stack in the case where the same variable name is re-used (possibly from the same place within two nested invocations of the same defined predicate) within a scope in which it has already been

defined. In each of its appearances, such a name may be bound to a different attribute-id; in such cases, the most recently added stack frame (the frame on the "top" of the stack) is treated as "current".

The processing of existentially quantified formulæ is now quite simple to describe. Upon entering such a formula, LISEC first creates a new stack frame containing information about each of the existentially quantified variables introduced in the current formula. Next, the current ADE is (recursively) constrained by its body, which is implicitly treated as a conjunction of one or more subformulæ (see Subsection 2.5.4). Upon exit from the existentially quantified formula, each of these existentially quantified variables is *projected* out of each term in the resulting ADE for reasons of efficiency. Readers familiar with Codd's constructive completeness proof [Codd, 1972] may notice that our maintenance of a logical variable stack serves what may be regarded as a dynamic analogue of the process of conversion to prenex normal form, but adapted to the case where the required renaming operations cannot be determined statically on a purely lexical basis.

2.5.3 Universal quantification

As in the case of the existential formula, processing of universally quantified formulæ begins with the creation of new attribute-ids to identify each of the newly-introduced universally quantified variables, and a new stack frame is created to record the type ("attribute id-valued") and value (the newly generated unique attribute-id name) of each such variable. The substantive portion of the processing of universally quantified formulæ within LSEC, however, is considerably more complex than the corresponding part of the procedure for processing existentially quantified formulæ. As noted in Subsection 2.4.7, all such formulæ are of the form

$$\forall x\, Q(x) \supset R(x) \quad ,$$

where the *qualification clause* $Q(x)$ is restricted to contain no disjunction or universal quantification. This qualification clause is used to restrict the range of the universally quantified variable x in each of the possible *contexts* defined by alternative joint instantiations of the various quantified variables that are "visible" within the current scope. We now consider in some detail the manner in which LSEC constrains the current ADE by a universally quantified formula.

Recall first that the ADE is in general a set of *several* relations, called the *terms* of that ADE. LSEC constrains each term by the universal formula, then takes the union of the results to form a new ADE. In order to constrain a given extension term by a universal formula, LSEC must first identify a crucial set of variables called the *context variables* of that formula. The context variables are precisely those attribute-id-valued variables that appear within the qualification clause, but whose innermost scope is not local to the qualification clause. (Our somewhat unwieldy definition is required to insure

that a variable appearing within the qualification clause at a point where its name is bound both locally – that is, within the qualification clause – and globally is not treated as a context variable.

LSEC next computes what is called the *qualified extension term* by (recursively) constraining the extension term by the qualification clause ($Q(x)$) of the universal formula. The qualified extension term is then *projected* over the attributes corresponding to each of the context variables (as determined by the current stack bindings for those context variables), yielding a set of context *tuples*. Each such tuple defines one *context* for evaluation of the body of the universally quantified formula. In intuitive terms, a context may be thought of as containing all relevant information about one possible way in which the qualification clause might be satisfied – that is, one "such that ..." condition in a universal formula which might be described by the English assertion "for all x such that $Q(x)$ is true, $R(x)$ must also be true". Note that it is not sufficient in general to find all x satisfying this "such that ..." clause, and to simply substitute all such x into the body of the universal formula to obtain a new set of constraining formulæ. In order to avoid excluding joint instantiations which which might well satisfy the top-level query, information regarding constraints on variables *global* to the universal formula must be propagated along with each such x value.

Each of these context tuples will ultimately contribute to the result for the current extension term, the partial results due to each being appended together at the end to form a new extension. Let us now consider the manner in which a given context tuple is used to constrain the qualified extension term by the body of the universally quantified variable. First, the qualified extension term is *selected* on the attributes and values specified in the context variable list and the current context tuple, respectively, to obtain the *context-*

bound extension slice for that context tuple. By projecting the context-bound extension slice over the universally quantified variable, it is now possible to identify the effective range of the universally quantified variable within the current context, resulting in a list of *context-bound universally quantified values*.

To obtain the partial result, the universally quantified variable is first projected out of the qualified extension term slice. The result (corresponding to the current context tuple) is then constrained by various instantiated versions of the body in succession -- first, by a version of the body with the first context-bound universally quantified value substituted for the universally quantified variable, next with the second such value substituted for that variable, and so on for each of the possible values within the effective (context-bound) range of the universally quantified variable. The result corresponding to this context tuple is now combined with those derived from each of the others to obtain a version of the original extension term constrained by the universally quantified formula. As noted above, the final result is obtained by taking the union of the results due to each extension slice in the original ADE.

As in the case of existential quantification, the universally quantified variables are projected out of the result upon leaving the scope of the universally quantified formula for reasons of efficiency.

2.5.4 Conjunction

It is in the case of a conjunctive formula that the process of progressive constraint which underlies the operation of the LSEC algorithm is most evident. Upon encountering a set of conjoined subformulae

$$P(x) \wedge Q(x) \wedge R(x) \wedge \ldots ,$$

the old ADE is first constrained by $P(x)$, the result is then further constrained by $Q(x)$, then by $R(x)$, and so on until either the extension has been constrained by the last conjoined subformula or a *false-extension* is returned by one such constraining step, in which case computation terminates with the value *false-extension*.

2.5.5 Disjunction

In the course of constraining the ADE by a disjunctive formula, the "width" of the ADE -- more precisely, the number of relations which it comprises -- is, in the general case, increased to reflect a larger number of alternative ways in which the query might be satisfied. To constrain the current ADE by a disjunction

$$P(x) \vee Q(x) \quad ,$$

for example, two copies of the ADE are made; one is constrained by $P(x)$, the other by $Q(x)$, and the results combined to form the new ADE.

2.5.6 Defined predicates

The processing of a defined predicate within the LISEC algorithm is analogous to the binding of λ-variables within LISP. A new stack frame is created to associate with the formal parameters all relevant information inherited from the actual parameters. At the time of binding, each formal parameter is designated as either a *constant-valued* variable, if the corresponding actual parameter is either a constant or has itself already been classified as a constant-valued variable, or an *attribute-id-valued* variable, in the case where the corresponding actual parameter is itself attribute id-valued (by virtue of having been bound at some level to a quantified variable).

Following creation of the new stack frame, the current ADE is (recursively) constrained by the body of the defined predicate, with its current bindings. For syntactic simplicity, the body of a defined predicate may be a list of conjoined subformulæ, and need not be an explicit conjunction. Typically, then, a defined predicate is processed by establishing new bindings and then evaluating the list of conjoined subformulæ which make up its body.

2.5.7 Primitive predicates

It is in the processing of primitive predicates that most of the computational effort of the LSEC algorithm is expended. For a simple illustration of the computationally demanding aspects of this process, consider the processing of the simple query

$$(x, y) : \exists z \; P(x, z) \wedge Q(z, y) \quad ,$$

where P and Q are both primitive predicates.

Since the body of the query is a conjunction, the initial ADE (*true-extension*) is first constrained by the primitive predicate $P(x, z)$. Constraint of the *true-extension* by a primitive predicate is treated as a special case by the LSEC algorithm. The resulting ADE contains a single relation, the *independent extension* of the primitive predicate in question. The independent extension of a primitive predicate is defined as the result of *selecting* the corresponding primitive relation in the extensional database on the values of any constant-valued arguments that may be specified, then *projecting* out the attributes corresponding to all such constant-valued arguments. In our example, P has no constant-valued arguments; the result is thus the degenerate case of an independent extension: a new ADE consisting of the single primitive relation corresponding to P.

This ADE is next constrained by the second conjoined factor, the primitive predicate $Q(z, y)$. In this more typical case, where the ADE is not equal to *true-extension*, the independent extension of $Q(z, y)$ is first computed by selection and projection of the primitive relation corresponding to Q, as described above, and then *joined* with the current ADE over all common attributes. In our example, the independent extension of $Q(z, y)$ would be joined with the old ADE (the independent extension of $P(x, z)$) over the common existentially

quantified variable z. In our knowledge-based retrieval task, this join operation, which would in general be very expensive on an ordinary machine in the case where the relations involved are of large cardinality, occurs quite frequently in the course of executing the LSEC algorithm, and would probably account for most of the execution time in a realistic application. It is the need to perform such joins in a highly efficient manner which thus provides what is probably the most important justification for the design of the database machine that will be described in Chapter 3.

2.5.8 The *result* formula

Up to this point, we have considered only the treatment of existentially and universally quantified variables by the LSEC algorithm. It will be recalled, however, that the "top-level" query formula must always contain at least one, and possibly more, *free variables*, all possible satisfying combinations of which will ultimately be returned as the *result* of the query. During the bulk of the LSEC algorithm, free variables are treated in exactly the same manner as existentially quantified variables: distinct attribute-ids are created for each, and the associated information stored on the logical variable stack without any indication of their special status.

After the initial ADE (*true-extension*) has been constrained by the full well-formed formula, however, each of the relations in the resulting ADE is *projected* over the result variables, and the union of the (necessarily union-compatible -- see Subsection 2.4.4) resulting relations is taken to yield the query result. In our demonstration system, for example, each relation in the final ADE is projected over the attribute-id corresponding to the top-level target document descriptions, and the union of the resulting unary relations—a new relation listing each of the matching targets—is displayed to the user.

2.5.9 On the complexity of LSEC

As we have already noted, our proposed database machine is designed to execute the primitive operations of the LSEC algorithm algorithm in a highly efficient manner. Since a number of these relational algebraic operations will in general be required in the course of an actual retrieval task, however, it is reasonable at this point to consider the complexity of the LSEC algorithm *in terms of* these relational algebraic primitives. First, it should be noted that the individual who constructs the set of defined predicates (which, in our thesis system, implement the match semantics) may exercise a considerable degree of explicit control over the sequence of operations that will ultimately be performed in the course of executing the LSEC algorithm. In practice, it has been our experience that predicate definition is an activity more nearly like ordinary (albeit very high-level) programming than, say, the analogous task confronting the architect of a resolution theorem proving system. In particular, it is possible to define two "weakly equivalent" sets of predicates—that is, two sets which are indistinguishable on the basis of their input/output behavior under the LSEC algorithm—such that one is considerably more efficient than the other.

It has been our experience that the number of relational algebraic operations which occur in the course of retrieving target descriptions matching pattern descriptions of realistic size and complexity is fairly modest (no more than a few dozen such evaluations for the most detailed of our test descriptions, for example). To be sure, the number of such operations could in theory grow quite rapidly as the size of the pattern description grew very large—the exact behavior depending both on intrinsic characteristics of the high-level description language and on factors under the control of the individual responsible

for predicate definition. In practice, however, the issue of query size and complexity is much less important than that of database size, particularly in the case of the very large databases to which our thesis research is directed. In this regard, it is the fact that the number of relational algebraic operations, while directly related to query complexity, is *independent* of the size of the database, which is of central concern. The critical determinant of system behavior in realistic large-scale database applications is thus the efficiency with which the individual relational algebraic primitives—particularly the join operator, by virtue of its complexity and frequency of invocation within I.SEC—are performed on the relational database machine. This issue forms the central focus of the next chapter.

CHAPTER THREE

THE RELATIONAL DATABASE MACHINE

In this chapter, we will describe the architecture of our proposed database machine, and will present and analyze a set of algorithms for the highly efficient evaluation of the relational algebraic primitives introduced in Chapter 2 on this machine. The proposed architecture is organized as a two-level hierarchy of associative storage devices- the smaller and faster level of which will be called *primary*, and the larger and slower, *secondary*. In the course of evaluating each relational primitive, the entire database is associatively probed using logic associated with the secondary storage devices themselves. In the case of all but two of the operators, selected segments of the relevant data are then transferred in succession from secondary to primary associative storage for further processing. This chapter will outline the organization of the architecture we are proposing, and will describe and analyze algorithms for evaluation of each of the relational algebraic primitives, both in the case where the arguments can fit entirely within primary storage (which we call *internal evaluation*), and where they reside on secondary storage (*external evaluation*).

In Section 3.1, we survey previous work on the design of associative processors and database machines to provide a context for the introduction of our architecture. Section 3.2 provides a functional description of the central hardware components involved in our design. The notation to be used in analyzing the performance of our algorithms for evaluation of the primitive relational operators is introduced in Section 3.3. The algorithms for internal evaluation are presented in Section 3.4, along with an average-case analysis of their time complexity. The procedures for external evaluation (most of which make use of the internal evaluation routines) are described in Section 3.5. In Section 3.6, two alternative schemes are described for the partitioning of the argument relations into appropriate segments and their transfer into primary storage, a time-critical part of the process of external evaluation. The latter of these two

schemes, which would appear to be preferable when appropriate hardware is available, is analyzed in some detail. Our results are summarized in Section 3.7; the reader may wish to glance briefly at this section before proceeding with the remainder of the chapter.

3.1 Relevant Previous Work

In this section, we will briefly survey certain areas in the literature of computer architecture that have central relevance to our own investigations. Because the great majority of the recent work on specialized architectures for database management applications—our own included—has drawn heavily on earlier work involving the design of content-addressable memories and associative processors, we will begin our review, in Section 3.1.1, with a rough taxonomy and description of the most important classes of associative processing devices. In Section 3.1.2, we will consider several of the best known proposals for, and actual prototype implementations of, what might be called true *database machines*: systems oriented toward fairly specific functions deemed relevant to actual database management applications.

3.1.1 Associative processors

At the risk of oversimplification, it is probably safe to say that virtually all existing and proposed database machine architectures have drawn their power from the utilization of a high degree of some variety of *hardware parallelism* at some level within the system. The different techniques for achieving such parallel computation are often distinguished according to a classification scheme suggested by Flynn [1972], which characterized the conventional sequential processor as a *Single Instruction Stream Single Data Stream* (SISD) machine, as contrasted with the most common mechanisms for parallel computation, among which he distinguished three different organizations:

1. *Multiple Instruction Stream Single Data Stream* (MISD) machines, typified by the *pipeline processing* approach.
2. *Single Instruction Stream Multiple Data Stream* (SIMD) machines, in which a single operation is performed in parallel by a number of independent processing units at any given time.
3. *Multiple Instruction Stream Multiple Data Stream* (MIMD) machines, which function as a number of independent, but communicating processors, each of which is capable of maintaining its own instruction and data streams.

Alternative classificatory schemes for parallel machines have also been suggested (eg., Murtha and Beadles [1964]). For a more thorough discussion of the taxonomy of parallel processors in general, the reader is referred to Thurber and Wald [1975].

Within the class of SIMD machines, two important subclasses are typically recognized. (Again, however, other classifications are possible; see, for example, Higbie [1973].) Members of the first subclass, exemplified by such machines as ILLIAC IV, are known as *array processors*, in which the data are

processed in parallel using the conventional mechanism of *coordinate addressing*. The second subclass consists of the associative *processors*, which access their data in a *content addressable* manner. Although each of the varieties of parallel processing which we have described above may ultimately play an important role within practical database machines, our primary concern in this dissertation will be with the family of associative processors.

In general, we will define an associative processor to be a machine which is able to access selected items stored in memory in parallel on the basis of their contents. We will also require that items be accessible by *partial match*, so that selected elements of the "key" field can be "masked out" in the course of content-based addressing. (In many associative processors, the match criteria may be specified in using predicates other than equality—arithmetic comparison operators, for example; we will not require this capability as part of our definition of an associative processor, however.) While the parallel modification of content-selected responders is supported directly by a hardware *multiwrite* capability, output from an associative processor in the case where there is more than one responder presents a more complicated problem. A number of designs have been proposed and evaluated for reading out a single responder in the event of a multiple match. Typically (though not always) this responder is chosen arbitrarily on the basis of physical position within the associative memory. Although much work has been done in the area of *multiple match resolution*, we will not be concerned with such problems in this thesis.

While the distribution of intelligence among memory elements is central to the operation of all associative processors, the degree of that distribution—more specifically, the number of storage elements associated with each comparison logic unit—varies widely among the various classes of associative

devices. In the remainder of this section, we will review the major categories of associative processor architecture, distinguished by the extent of distribution of the processing logic. A survey by Yau and Fung [1977] provides an outline of associative processor architecture in somewhat more depth than will be possible here. The area is treated even more thoroughly in an outstanding book by Foster [1976].

The greatest degree of distribution is found in the *fully parallel* associative processors, in which comparison logic is associated with every bit of storage. The fastest of these machines are the *fully parallel word organized* associative processors, whose hardware complexity, however, has resulted in their implementation only experimentally, and on a very small scale.

A second class of fully parallel designs is represented by the *distributed logic* associative processors. In the original distributed logic associative processor design introduced by Lee [1962], one comparison logic unit is associated with each character of storage. (In some variants, the comparison logic unit is instead associated with a small, fixed-size set of adjacent characters.) In all of the distributed logic associative processors based on Lee's design, each cell is capable of storing a small amount of "state" information in addition to the symbol data itself. The design includes a control unit that communicates with all cells in parallel using a common databus. Each cell, however, is connected not only to this public bus, but also to its immediate right and left neighbors, thus forming a *rail* along which control and state information can be propagated.

With some simplification and disregard for detail, a string of data stored in a contiguous set of character cells is retrieved as follows. Initially, the control system "broadcasts" a special "word header" character which precedes all strings stored in memory. Each matching header cell is then instructed

to enable its right neighbor for comparison against the first character in the search string. All matching first characters in turn enable their right neighbors for matching against the next character, and so on until the search string is exhausted. The set of matching strings is now easily identified, and may be modified or output. A number of variations on Lee's original distributed logic design have been proposed to deal efficiently with certain operations frequently required in the course of information retrieval, parallel arithmetic manipulations, etc. (eg., Lee and Paull [1963]; Gaines and Lee [1965]; Crane and Githens [1965]). The content addressing mechanisms incorporated in PEPE, one of the first large scale associative processor implementations, may also be regarded as derivative of Lee's design.

Among the numerous distributed logic designs which have been suggested, the Tree Channel Processor architecture proposed by Lipovski [1969; 1970] for the construction of very large primary associative processors is worthy of special mention. In Lipovski's design, the cells are themselves capable not only of passive comparison and simple propagation, but (in a particular mode of operation) of the active execution of certain control functions. The cells are organized in a tree structure, with "adjacent" cells connected by two separate rails and a "locally" common channel. In contrast to the strictly public bus used in the basic distributed logic design, this channel may be dynamically partitioned, thus isolating one or more subtrees which can then function as separate processing units. (In this respect, the Tree Channel Processor might in fact be considered an unconventional MIMD machine.) The Tree Channel Processor is designed to permit extremely high bandwidth parallel input and output and to greatly reduce certain propagation time bottlenecks associated with many applications of distributed logic processors.

Let us now turn our attention to a class of associative processors characterized by somewhat less extensive distribution of intelligence: the *bit-serial* associative processors. In this class of machines, first proposed by Shooman [1960], the content addressable memory is organized into (often fairly large) words, and comparison logic is associated with each word. In contrast with the fully parallel word organized processors, however, each logic unit is capable of manipulating only a single bit position within the word at a given time, resulting in a reduction in required processor logic roughly proportional to the number of bits per word, at the cost of a corresponding decrease in speed. At each point in time, one "bit slice" extending through all words is thus accessible for parallel processing. A small amount of storage associated with each word is typically used to retain state information between operations on successive bit slices in support of the primitive content search and multi-write capabilities of associative processing.

Since the introduction of Shooman's "orthogonal computer", bit-serial associative devices having a wide variety of characteristics have been proposed by a number of researchers, including Kaplan [1963], Ewing and Davies [1964] and Chu [1965]. The design of STARAN [Rudolph, 1972; Batcher, 1974], Goodyear Aerospace Corporation's large-scale associative processor, is based on a group of "multidimensional access memories" which implement both bit-slice (for associative processing) and ordinary word slice (for input and output) access capabilities using standard random access memory chips. Among the other bit-serial associative processors which have been developed for practical use are the OMEN series [Higbie, 1972], designed by Sanders Associates, The Raytheon Associative/Array Processor (abbreviated RAP, but not to be confused with the Relational Associative Processor, a database machine bearing the same acronym which will be discussed in Section 3.1.2) [Courant, Gerhardt and Young, 1974], the Extended Content Addressed Memory (ECAM) [Anderson

and Kain, 1976], and the Hughes Aircraft Associative Linear Array Processor, (ALAP) [Finnila, 1977]

Another class of less-than-fully-parallel content-addressable devices is comprised of the *word-serial* associative processors [Young, 1962; Crofut and Sottile, 1966, Rux, 1969], in which all bits of a *single* word are compared in parallel, but the set of words is examined sequentially. Word-serial machines thus function in much the same way as a program loop on a conventional von Neumann machine in which each word in memory is examined in turn for partial match and modified or output as appropriate. The word-serial associative architecture, however, obviates the need to fetch and decode the instructions which would be required to perform such functions in software on an ordinary von Neumann machine. While the word size of a word-serial machine might in principle be chosen large enough to make word-serial techniques competitive in speed with bit serial schemes, the number of words is generally much larger than the number of bits per word, thus typically making word-serial techniques much slower in practice. Although this speed disadvantage has thus far tended to discourage practical applications of the word-serial approach in favor of distributed logic and bit-serial techniques, current prospects for inexpensive, high density, noninertial circulating storage devices (future generations of bubble memories or charge coupled devices, for example) may make the word-serial approach worthy of serious consideration for large-scale associative processing applications.

At the low end of the associative logic distribution spectrum is the class of *block-oriented*, or *segment sequential* associative processors (also sometimes called *partially associative* devices), which offer much larger capacities than the devices discussed thus far, but at a significant penalty in speed. Most commonly, such devices are based on a rotating storage device (a disk, for

example) having one or more heads per track of storage, so that each piece of stored information passes under some head exactly once during each revolution of the device. In the simplest such designs, one search and modification logic unit is associated with each head (and thus with each track), permitting one associative operation to be performed on each revolution.

The first block-oriented associative devices of which we are aware were proposed by Slotnick [1970] and Parker [1971]. Parhami [1972] designed an associative processor called RAPID (for Rotating Associative Processor for Information Dissemination), which functioned in much the same way as a slow distributed logic memory, but with only one search operation possible per revolution, and with information propagation in one direction only. Different block-oriented associative processor designs have been proposed by Minsky [1972], Healy, Lipovski and Doty [1972], and others. Because the need for large-scale storage is essential to data base management applications, parallel head per-track disk devices, or their equivalent, have a central role in the majority of the database machine designs discussed in Section 3.1.2.

While the block oriented associative processors are usually regarded as representing the "low end" of the logic distribution spectrum within the family of associative processor architectures, certain system designs based on an even lower degree of distribution, but nonetheless sharing some of the features of an orthodox associative processor, might be worth mentioning in passing. One such approach is illustrated by the modified head-per-track disk drives incorporated in the DBC architecture (discussed in Section 3.1.2), in which the contents of one cylinder may be associatively probed in a single revolution. An even lower degree of logic distribution which nonetheless speaks to some of the concerns of associative processing is represented by the design proposed by Lang, et al. [1977] for the evolutionary enhancement of conventional disk-

based systems the authors' proposal was in fact presented in the context of an architecture like that of the IBM System/370 for increased efficiency in database applications. Their scheme involved the association of one small, intelligent unit called a "DASD processor" with each direct access storage device. Each such processor would be capable of searching for records based on the values in arbitrarily specified (as opposed to fixed position) fields when so instructed by a special channel command. Analysis predicted significant performance improvement over more conventional system architectures, particularly in the case of heavy transaction traffic.

In the following subsection, we will review some of the ways in which the various classes of associative processors have been applied to the specific problems of database management.

3.1.2 Database machines

Several authors have surveyed the emerging field of database machine architecture from various perspectives, and adopting various scopes, within the past several years. Linde, Gates, and Peng [1973] were among the first to point out the potential advantages of associative processor-based architectures for real-time database management applications. Berra [1974] reviewed the state of the art as of 1974, and critically examined the potential for such applications, pointing out a number of positive and negative aspects of the application of associative processors to database management. Anderson [1976] and Baum and Hsiao [1976] provided later overviews of trends in the field, the latter predicting the emergence of hierarchically organized systems, with each level containing functionally specialized search and data manipulation modules. Lowenthal [1977] offered a taxonomy for distinguishing three different kinds of processors specialized for data base management in distributed environments, which he called *intelligent controllers*, *backends* and *datacomputers*. Hsiao and Madnick [1977] and Berra [1977] also survey and provide references to the field of data base machine architecture. In this subsection, we will review the best known efforts to date in the area of database machine architecture.

One of the earliest actual implementations of an associative processor-based system geared toward database management applications was IFAM (DeFiore and Berra [1973]), developed on an experimental prototype basis for the Rome Air Development Center. This implementation of IFAM was based on a 2048-word, 48-bit, word parallel, bit serial associative processor called AM, developed by Goodyear. The capabilities of IFAM were closely tied to the primitive associative operations, in relational terms, tuples could be retrieved only by selection (although with inequality and "within-range" comparisons

in addition to simple equality). Although limited in storage capacity by comparison to later block-oriented associative processor-based database machines, IFAM served to concretely illustrate the potential utility of associative operations in database management applications.

Moulder [1973] described an implementation of a hierarchical database management system based on STARAN (described in Subsection 3.1.1) and a parallel head per-track disk drive. Using a technique similar to that described by DeFiore, Stillman and Berra [1971], the hierarchical data structures chosen for data representation were converted into a single level data base to permit the use of the associative processing capacities of the hardware. Retrieval was again by selection based on equality or inequality (but not ranges) over various attributes. The database was partitioned into a number of physical disk sectors which were sucessively read into the STARAN memory arrays in a high speed parallel fashion, where they were searched using the associative capabilities of STARAN. The time required in the case of typical queries to perform these associative searches within the STARAN arrays was found to be small enough that every other sector could be searched in the course of one revolution of the disk, so that the whole data base could be searched in two revolutions (about 78 msec in the prototype system).

One of the first large-scale research efforts directed toward the development of a specialized system containing many of the features critical to database management is represented by the CASSM project, active at the University of Florida since 1972. CASSM [Su, Copeland and Lipovski, 1973; Copeland, Lipovski and Su, 1973; Lipovski, 1978] is a block-oriented design oriented specifically toward a hierarchical data model, providing a direct hardware implementation of hierarchical data structures, which are linearized in a top-down, left-to-right manner. CASSM is capable of supporting the relational

and network (Codasyl DBTG) models as well, however.

In the terminology of CASSM, the system architecture includes a collection of identical *cells*, each consisting of a processing element and a circulating sequential memory element (a disk track or a circularly organized CCD or bubble memory device, for example). Each processing element can communicate with its two immediate neighbors, in support of the storage of files and records which overlap physical segments of the secondary storage device. Associated with each cell are two heads: one used for reading, and one for writing data. After being read by the first head, data is pipelined through a chain of processing logic, each portion of which serves a specialized function. CASSM includes special features for searching complex data structures such as sets, ordered sets, trees, variable length character strings and directed graphs. Among the distinctive features of CASSM is the fact that both programs and data are stored on the associative secondary storage device. Both an assembly language [Su, Chen and Emam, 1978] and a high-level nonprocedural language [Su and Emam, 1978] have been developed for programming the CASSM system. A single cell prototype system was completed in 1976. Since that time, efforts have concentrated on the implementation of and experimentation with a software simulation of a multi-cell CASSM system.

The best-known database machine designed specifically for efficient support of the relational model of data is probably RAP (for Relational Associative Processor), developed at the University of Toronto [Ozkarahan, Schuster and Smith, 1974, 1975; Schuster, Ozkarahan and Smith, 1976; Ozkarahan, 1976]. RAP is designed as a backend database processor for a general purpose computer, accepting from the latter a set of primitive commands relevant to the evaluation of relational queries. Like CASSM, the RAP architecture is organized around a set of identical cells, each consisting of a processor and a

block of circulating memory, and capable of various retrieval, insertion, deletion and update functions. All cells are connected to a common controller, which includes a statistical arithmetic unit. Simple inter-cell communication facilities are provided for priority polling in the course of output. The front end computer is used to translate different query languages into RAP primitives, to handle various input/output processes, for query scheduling, and for various functions related to the maintenance of protection, security and data integrity. The RAP language interface is described by Kerschberg, Ozkarahan and Pacheco [1976] and Ozkarahan and Schuster [1976].

An analytical comparative performance evaluation [Ozkarahan, Schuster and Sevcik, 1977] revealed advantages in speed ranging between one and three orders of magnitude by comparison with a hypothetical conventional system using inverted lists – with the very important exception of the join operation, where only a slight improvement was found. (Note that it is just this sort of operation for which our own architecture offers the greatest potential advantages.) Specific aspects of the performance of of RAP are examined by Nakano [1976] and Ozkarahan, Schuster and Sevcik [1977]. The RAP system has now evolved for several years, with the latest version, called RAP 2 [Schuster, Nguyen, Ozkarahan, and Smith, 1979], embodying several significant changes. First, a general purpose microprocessor has been employed for implementation of the previously hardwired controller. Second, the RAP 2 design is strongly oriented toward the use of CCD memories instead of head-per-track disk devices. The instruction set has also been modified somewhat in RAP 2 to make it more uniform and flexible, and to add certain additional capabilities. Enhancements based on analogues of multiprogramming and virtual memory organizations have been proposed by Ozkarahan and Sevcik [1977].

Another architecture specifically oriented toward the relational database

model is embodied in a proposed database machine called RARES [Lin and Smith, 1975; Lin, Smith and Smith, 1976]. The RARES design is distinguished primarily by the adoption of an *orthogonal storage* layout, in which individual tuples are distributed *across* (and not *along*) the tracks of the parallel head-per-track secondary storage device, with one byte stored on each track. In the orthogonal storage scheme, a given relation thus occupies all tracks within a particular sector of the disk device (whose extent depends on the size of the relation), rather than completely filling a corresponding number of tracks. One motivation for the orthogonal scheme adopted in the RARES design is to reduce the incidence of contention in cases where more than one tuple is identified in parallel for output. Among the other advantages cited for this scheme are a reduction in the amount of storage necessary to hold each tuple in the course of associative comparison and certain efficiencies in the execution of operations on relations in which a sorted order must be maintained.

The relational database machine architectures we have thus far considered have primarily addressed the problems of evaluating a single relational primitive operation. An organization called DIRECT [DeWitt, 1979], on the other hand, is directed to a broader set of problems, dealing with such questions as intra- and inter-query concurrency and database integrity in a multiple-process relational database environment. DIRECT is a virtual-memory, MIMD (see Subsection 3.1.1) system, currently being implemented using a number of DEC LSI-11/03 microprocessors, along with CCD-based associative storage units. The microprocessors and CCD modules are connected using a special cross-point switch design, with the number of processors assigned to the evaluation of a given query determined dynamically based on certain statistics of the query and the relations involved.

At Ohio State University, an architecture has been proposed for a very-

large-scale database system based on the use of a number of interconnected subsystems specialized for different aspects of the process of database management. This system, called DBC (for database computer) [Baum and Hsiao, 1976; Hsiao, 1977, Banerjee, Baum, Hsiao and Kannan, 1979], is designed to support all three data models, communicating with a general purpose computer through a very high level language oriented toward the data base management functions for which DBC is intended. The design of DBC was strongly influenced by several kinds of data protection concerns, and includes specialized mechanisms for the imposition of related constraints.

The system is composed of two sets of processor and memory components, configured as closed loops, and interconnected (both to each other and to the general purpose computer to which DBC is subordinated) by a *database command and control processor*. The first, called the *data loop*, contains a *mass memory* based on a number of modified *moving-head* disk drives, along with a specialized processing unit called the *security filter processor*. The second, called the *structure loop*, is comprised of a block-oriented associative storage unit (envisioned to be constructed using CCD or bubble technology) called the *structure memory*, another specialized processing unit called the *structure memory information processor*, and two other specialized modules called the *keyword transformation unit* and the *index translation unit*.

The moving-head disk drives are modified to provide for simultaneous output from all tracks in a given cylinder in parallel. (Such drives have in fact recently been announced by Ampex Corporation [1978], and are apparently not expected to be priced far above the cost of unmodified moving-head drives.) Associated with each track is a *track information processor*, capable of associative comparison operations. Thus, a single cylinder can be searched associatively by DBC in much the same way as were the full contents of secondary

storage in the associative head per track devices discussed earlier. Information used to locate the relevant cylinders to search is stored in the structure memory unit, which is designed for very fast access and processing by the structure memory information processor, in conjunction with the keyword transformation and index translation units. A more detailed description of the structure memory, structure memory information processor, keyword transformation unit and index translation unit are provided by Hsiao and Kannan [1976] and Hsiao, Kannan and Kerr [1977]. The design of the mass memory, security filter and associated units are detailed in Hsiao and Kannan [1976a].

Other proposals for specialized database architectures include XDMS [Canady, et al., 1974] a network-oriented SISD architecture originating at Bell Laboratories, and an approach to the implementation of a relational database system suggested by McGregor, Thompson and Dawson [1976].

3.2 The Proposed Architecture

As noted in the introduction, our proposed architecture is configured as a hierarchy of associative storage devices. At the top of this hierarchy is the *primary associative memory* (PAM), a fairly fast content-addressable memory of relatively limited capacity. (For concreteness, the reader might imagine a PAM containing between 10K and 1M bytes, and requiring somewhere between 100 nanoseconds and 10 microseconds per associative probe.) PAM might be realized with a large-scale distributed logic memory, or with a suitable bit-serial or word serial design. There is reason to believe that recent progress in distributed logic architectures, device level fault-tolerant designs and wafer-scale integration could soon make such a memory unit feasible for wide application.

Two primitive PAM operations, each requiring a single associative probe, will be involved in our analysis: *mark all* and *retrieve and mark first*. In both cases, all tuples of a specified relation for which the value of a selected compound attribute is found equal to a particular constant are associatively identified. The *mark all* operation writes a one or zero in a specified *flag bit* of each such matching tuple using a parallel hardware multiwrite. The *retrieve and mark first* operation sets a specified flag bit within a single tuple chosen arbitrarily from among the responders and copies the value of that tuple to storage external to PAM, but accessible to the controlling processor.

As an alternative to physical content-addressability, the algorithms we will describe could be modified to accomodate a "pseudo associative" PAM, constructed using, say, random access memory and high-bandwidth special purpose hash coding hardware. In general, however, argument and intermediate result relations would have to be re-hashed (on different attributes) prior to every algebraic evaluation, adding a significant (and less predictable, as seen

from the wide discrepancy between average and worst case hashing behavior) amount of time to the algorithms presented in Chapter 6.

The *secondary associative memory* (SAM) is intended to be a larger, slower content-addressable device. (A capacity of between 1 and 100M bytes and an associative operation time of between 1 and 100 milliseconds should adequately exemplify our design.) Physically, SAM might be realized using an intelligent circulating storage device such as a parallel head-per-track disk with a modest amount of logic associated with each track, or a non-inertial circulating storage device constructed using CCD or bubble memory storage technology, and having similar logic associated with each storage loop. (The ability to temporarily suspend circulation in individual storage loops in the latter class of device could in fact be utilized to improve somewhat on the external evaluation results reported in this chapter, although such enhancements are not within the scope of our present discussion.)

It is assumed that the relative speeds of PAM and SAM are such that a quantity of data sufficient to fill PAM can be transferred from PAM to SAM in the course of a single SAM revolution. Although the combined potential bandwidth of the set of intelligent heads associated with the SAM device could in principle be extremely high, the average bandwidth in the course of a external evaluation will ordinarily be much lower. (In fact, the analysis presented in Section 3.6.2 provides a rigorous statistical bound on these bandwidth requirements under a relatively weak set of assumptions regarding the distribution of data values in the argument relations.) Given adequate buffering capabilities, the algorithms we will describe should thus present no unusually stringent requirements on the communication channel between SAM and PAM.

Among the specific capabilities assumed for the "per-track" logic of an acceptable SAM device is the ability to output or mark all tuples for which

the values of selected attributes are found equal to a constant or to the value of some other attribute within that tuple, or to be within some specified range of values. Note that SAM is thus capable of evaluating the select and restrict operators directly, without recourse to "internal evaluation" within PAM. Each per-track logical unit is also assumed to have a sufficient quantity of random access buffer memory to hold the tuple currently passing under its head until a determination can be made, on the basis of selective or restrictive criteria, as to whether it satisfied the current match criteria.

The specifications we have thus far considered for SAM are quite similar to those of such actual rotating associative processors as those used in the RAP and CASSM systems. One of the techniques we will describe (in Subsection 3.6.1), however, also requires that each per-track processing unit have a small amount of random access memory dedicated to the tabulation of a "domain histogram". In addition, this algorithm requires that the unit be capable of determining whether each tuple satisfies one of a set of (not more than a small fixed number of) range specifications. In the alternative algorithm (described in Subsection 3.6.2), the per-track logic unit is instead assumed to have the capability of sequentially computing a hashing function on selected attribute values of each tuple which "passes under" the associated head (or its functional equivalent), and of outputting all tuples for which the resulting hashed value falls within a specified range.

The analytic portion of this chapter assumes a fixed time for an associative probe of the entire contents of SAM, as is the case for the sort of block-oriented associative processors discussed in Subsection 3.1.1 and employed in such database architectures as CASSM, RAP and RARES (Subsection 3.1.2). Our external evaluation algorithms are also applicable, however, to the kind of modified moving-head disk devices employed in the DBC design (Subsection

3.1.2), thus supporting very large data base applications. In order to adapt the complexity results reported in Section 3.5 to a SAM of this sort, in which only part of the database is associatively accessible on each rotation, a constant term would be added to the external evaluation times of each of the seven primitives. In addition, the complexity of these results would be increased (by a formally linear, although in practice probably quite small) factor in the event the argument relation(s) were allowed to exceed the capacity of the cylinder or cylinders capable of simultaneous parallel examination (at least in the absence of a significant modification of our algorithm). To simplify our discussion, however, the remainder of this chapter will assume that SAM is a fixed probe time associative device of sufficient capacity to store both argument relations.

The seven relational algebraic primitives with which we are concerned may be evaluated most quickly when the argument relation(s) can fit into PAM—the case we have referred to as *internal evaluation*. (Similarly, we will use the terms *internal projection*, *internal equi-join*, etc., to refer to the evaluation of specific relational operators in the case where their argument relation(s) fit entirely within SAM.) *External evaluation* is performed whenever the argument relation(s) fit in SAM, but not in PAM, and in most cases involves the reading into PAM of successive segments of the argument relation(s), each of which is (are) processed according to the corresponding internal evaluation algorithms. Note that this implies that each tuple of the argument relations is processed only once in primary storage, in contrast with the best currently known general techniques for the external evaluation of most of the algebraic primitives under consideration on a conventional non-associative system.

In addition to the two associative devices involved in our design, we assume the existence of a general purpose processor serving as a controller for the evaluation process, and responsible for the performance or delegation to

other specialized units of all collateral functions (input language translation, input/output control, etc) which would be involved in a practical implementation Adequate buffering would also be required at several points within the design we are proposing Although we will give little explicit attention to such issues in the present dissertation, it should be acknowledged that the detailed design of a useful realization of the architecture we propose would require careful consideration of the nature and capacities of these resources.

3.3 Notation

The following notation will be used in our analysis of the algorithms for the internal and external evaluation of the relational algebraic primitives:

Fixed system parameters:

P	Size in bytes of the primary associative memory (PAM)
S	Size in bytes of the secondary associative memory (SAM)
T_p	Time for an associative probe (returning one matching tuple) in PAM
T_r	Time for one revolution of SAM

Functions of the argument relation(s):

$c(R)$	cardinality of the relation R
$t(R)$	(fixed) size of the tuples of R in bytes
$d(A, R)$	number of distinct values of the (compound) attribute A in R
r	cardinality of the result relation

Because the quantity $P/t(R)$ (roughly speaking, the 'tuple capacity' of PAM) plays an important role in our analysis, we will also define a derived function $a(R)$ with this value.

It should be noted that r is being treated as an independent variable, although it is in fact determined by the composition of the argument relations There are several ways in which this functional dependence might have been explicitly embodied in our analysis if we had chosen to do so. We might have used, for example, a fixed value estimating the average number of occurrences of any given join attribute value, or for a more careful analysis, a particular statistical distribution of such values might have been assumed. While such an analysis might well help to identify certain interesting properties of the proposed algorithms when applied to argument relations having various

properties, we have chosen in the present analysis to forego the considerable added complexity involved in explicitly examining such relationships, treating the cardinality of the result relation as a constant and indicating verbally its relationship to the arguments where appropriate.

When there is no danger of confusion, we will sometimes omit the relation argument *R*.

3.4 Internal Evaluation

Our algorithms for internal evaluation of the project and join operators will be expressed in a hypothetical parallel programming language having a Pascal-like format, but extended to include four high-level associative processing primitives. The first is the parallel set command, used to set a specified flag to *true* in each tuple satisfying certain conditions; all flags are set in parallel using a single *mark all* operation, requiring one associative probe. (In each algorithm, all flags are initially assumed to be *false*. This command has the form

parallel set (*flag*) in all (*tuple variable*) of (*relation*) with (*conditions*) ,

where (*conditions*) is a boolean combination of predicates involving the variable (*tuple variable*). The format of the parallel clear command is identical to that of parallel set, but sets the specified flags to *false*.

The third associative processing primitive is the for each control structure, which has the form

for each (*tuple variable*) with (*conditions*) [set (*flag*) and] do (*statement*) ,

where the "set ... and" clause is optional. Unlike the parallel set and parallel clear statements, execution of a for each loop is sequential (although each *iteration* of the loop involves the performance of parallel associative probes). During each iteration, a single *retrieve and mark first* operation is performed, during which (*tuple variable*) is instantiated with an arbitrarily chosen tuple satisfying (*conditions*). If a "set ... and" clause is specified, the appropriate (*flag*) is set within this tuple; (*statement*), which may be either a simple

statement or a "begin ... end" block, and which may set flags affecting the value of (*conditions*), is then executed with the current binding of (*tuple variable*). Iteration terminates when no further tuples of the specified relation satisfy (*conditions*).

The final primitive is a conditional statement, which has the form

if [not] exists (*tuple variable*) with (*conditions*) [set (*flag*) and] do (*statement*)

where **not** is optional. This statement executes a *retrieve and mark first* operation, executing (*statement*) if *any* tuple satisfies (*conditions*) (or in the case where **not** is specified, if *no* tuple satisfies (*conditions*)).


```
procedure project(R, A),
    for each t of R
      with not flag do              (r + 1 probes)
          begin                     (r times)
          output t[A];
          parallel set flag         (r probes)
            in all t' of R
            with t'[A] = t[A];
          end;
```

Algorithm 3.1. Internal Project

3.4.1 Project

The procedure for internally projecting a relation R over a compound attribute A is detailed in Algorithm 3.1.

From the execution counts, it can be seen that internal projection requires time

$$(2r + 1)T_p$$

in addition to the time required to extract the projected compound attribute of, and output, each of the r result tuples, both non-associative functions which could be overlapped with the following associative probe. As noted in Subsection 2.4.4, projection can be quite expensive on a von Neumann machine, particularly in the case where the argument relation is large. The utility of the proposed architecture for the evaluation of the relational project operator thus lies not only in the fact that it requires time independent of the size of the the argument relation (being proportional only to the cardinality of the result relation, which can never be larger, and is often much smaller), but also that it implicitly eliminates the possibility of tuple duplication, obviating the need for sorting, for example, to remove redundant result tuples

```
procedure join(R1, R2, A1, A2);
    for each t1 of R1
      with not flag
      set flag and do                          (d(A1, R1) + 1 probes)
          begin                                (d(A1, R1) times)
          distribute(t1, R2, A1, A2);
          for each t'1 of R1
            with t'1|A1| = t1|A1|
              and not flag
            set flag and do                    (c(R1) probes)
                distribute(t'1, R2, A1, A2);   (c(R1) - d(A1, R1) times)
          end;

procedure distribute(t1, R2, A1, A2);
    begin                                      (c(R1) times)
    for each t2 of R2
      with t2|A2|    t1|A1|
        and not flag
      set flag and do                          (r + c(R1) probes)
          output (t1|A1| | t2|A2|);
    parallel clear flag
      in all t2 of R2
      with t2|A2| = t1|A1|;                    (c(R1) probes)
    end;
```

Algorithm 3.2. Internal Join

3.4.2 Join

Algorithm 3.2 computes the equi-join (or with a slight modification to the output instruction, the natural join) of relations R_1 and R_2 over the compound attributes A_1 and A_2, respectively.

Intuitively, the join algorithm functions as follows: First, an arbitrary R_1 tuple is retrieved and marked. The extended cartesian product of all (associatively retrieved) R_1 and R_2 tuples having the same value in their respective compound join attributes as this arbitrarily selected tuple is then output.

Another arbitrary R_1 tuple is then arbitrarily selected from among those which have not yet been processed, and the above procedure repeated until all R_1 tuples have been exhausted, at which point the equi-join is complete. The process of forming the extended cartesian product involves a nested-iteration over all matching R_1 (in the outer loop) and R_2 (in the subfunction *distribute*) tuples, each of which is retrieved in a fixed amount of time, without regard to its position in memory, by virtue of the content-addressibility of PAM. Excluding the time required for concatenation and output,

$$(r + 3c(R_1) + d(A_1, R_1) + 1)T_p$$

is required for internal evaluation of the join operator.

Note that the asymmetry of this algorithm with respect to the roles played by the two argument relations permits a (possibly quite significant) increase in efficiency in the case where the relative sizes of the two argument relations is known or inexpensively computable. Some of the existing designs that might be chosen for a particular physical implementation of PAM are in fact capable of providing, in a single associative operation, a count of the number of responders to an associative probe. When this capability is provided, the above algorithm may be preceded by two counting probes (on all R_1 and R_2 tuples) to determine the smaller relation. When such relative size information is available, R_1 should be chosen to be the smaller of the two relations in order to minimize the size of the $c(R_1)$ and $d(R_1)$ terms, since $d(R_1)$ would in practice be directly related (or at least not strongly inversely related) to $c(R_1)$. This observation is particularly significant in the common special case where the two argument relations are of very different sizes. As it happens, our external algorithm for the evaluation of two very large relations A and B admits the

possibility of assigning A segments to R_1 during some of the internal cycles, and B segments during others. At the cost of a very minor complication of the procedures for transfer from SAM into PAM, the algorithm can thus in some cases be made to perform more efficiently than would be the case if *either* A or B were "bound" to R_1 for the duration of the join, yielding a modest improvement on the above results

As in the case of projection, it is instructive to compare the proposed associative equi join algorithm to the best known general algorithms for this operation on a conventional von Neumann machine, which, as seen from the discussion in Subsection 2 4 4, are of $O(n \log n)$ complexity in the absence of physical clustering with respect to the join attributes or the use of extensive storage redundancy. On the machine we have described, on the other hand, tuples can be set in correspondence using a procedure of lower computational complexity than sorting, yielding a joining time which is *linear* in the cardinality of the smaller argument relation, the number of distinct join attribute values in this relation, and the size of the result relation (As we shall see in Subsection 3 5 3, linear complexity is preserved in the external algorithm for equi join as well)

Lest these results be misinterpreted, it should be emphasized the *worst case* behavior of this algorithm (or indeed, of any algorithm involving sequential output, regardless of the underlying architecture) may still be quite bad when the *result* relation is very large. Specifically, if for all $t_1 \in R_1$ and $t_2 \in R_2$,

$$t_1[A_1] = t_2[A_2] = t_c$$

for some single constant tuple t_c, the cardinality of the result relation will be equal to the product of the cardinalities of the two input relations Given

reasonable assumptions reflecting the typical use of the join operation, however, the architecture and algorithm we have described offer a very significant increase in efficiency.

It is worth noting that the algorithm we have described assumes access only to a *syntactic* model of the data, and not to any of the *semantic* characteristics of the stored relations (both terms being understood in the senses applied in the relational database literature). In fact, such semantic information, if available, could be used to significantly improve on several important special cases of the above join algorithm. As an example, consider the case where the compound join attribute is in fact a *primary key* of R_1, R_2 or both— that is, where the value of the join attribute uniquely identifies a single tuple of the relation. In this case, the associative probe used to terminate the above for each control structures is unnecessary, resulting in a saving of roughly half of the necessary probes within the innermost two loops of the algorithm In many problems, the availability of *domain-specific knowledge* might permit certain other kinds of improvements on these results Although an adequate analysis of the manner in which such additional sources of information might be profitably integrated into our approach is unfortunately beyond the scope of our present discussion, it is worth noting that the very general case of evaluation on the basis of purely syntactic characteristics, to which our attention is currently directed, may often in practice ignore information sources that might lead to increased efficiencies

```
procedure select(R, A, V);
    for each t of R
      with t[A] = V
        and not flag
      set flag and do              (r + 1 probes)
          output t;
```

Algorithm 3.3. Internal Select (with sequential output)

3.4.3 Select

The algorithm for selection is quite straightforward within the architecture we have specified, since the associative retrieval primitive that defines the behavior of PAM itself serves what is essentially a selective function. If, in a particular application, it is not necessary to sequentially enumerate and output the result of a selection, but only to *mark* the included tuples, (as may in fact be the case in the evaluation of many complex queries), the operator in fact requires only a single probe, and takes exactly time T_p, independent of the size of the argument relation. When sequential output is required, the selection from relation R with compound attribute A equal to value tuple V is defined as in Algorithm 3.3. It is easily seen that $r + 1$ probes are required, so that the time required for a single selection with sequential output is simply

$$(r + 1)T_p$$

It should be noted that the time required for selection with sequential output is again independent of the size of the argument relation.


```
procedure restrict(R, A1, A2);
    for each t of R
      with not flag do                 (d(A1, R1) + 1 probes)
          begin                        (d(A1, R1) times)
          for each t' of R
            with t'[A1] = t[A1]
              and t'[A2] = t[A1]
              and not flag
            set flag and do            (r + d(A1, R1) probes)
                output t'              (r times)
          parallel set flag
            in all t'' of R
            with t''[A1] = t[A1] of R  (d(A1, R1) probes)
          end;
```

Algorithm 3.4. Internal Restrict

3.4.4 Restrict

The procedure for internal restriction is detailed in Algorithm 3.4. Initially, an arbitrary tuple is chosen from the argument relation. If the A_1 and A_2 values of this tuple are equal, one tuple having this value for both A_1 and A_2 is output during each successive probe until exhaustion. At this point, *all* tuples having that value for their A_1 attribute are flagged, and the process is repeated on all unflagged tuples. The total time required for internal restriction is

$$(2d(A_1, R_1) + r)T_p$$

It is worth mentioning that the addition of certain hardware capabilities to the PAM device may substantially decrease the complexity of internal restriction. If the hardware permits the associative retrieval of all tuples in which a boolean disjunction of attribute-value pairs is specified, for example, the parallel set instruction can be changed to flag all tuples in which the value of either

A_1 or A_2 is equal to $t[A_1]$; the elimination of such tuples may exclude from consideration some of the $d(A_1, R_1)$ tuples having distinct A_1 values without the need for a separate associative probe in the outer loop. A more significant improvement may be possible if the PAM device itself supports the associative retrieval of tuples having identical values in specified fields; in this case, internal restriction has the same complexity as selection.


```
procedure union(R1, R2),
      begin
      for each t1 of R1
        with not flag
        set flag and do                          (c(R1) + 1 probes)
            begin                                (c(R1) times)
            output t1;
            parallel set flag                    (c(R1) probes)
              in all t2 of R2
              with t2 = t1
            end;
      for each t2 of R2
        with not flag        set flag and do     (r - c(R1) + 1 probes)
            output (t2;
      end;
```

Algorithm 3.5. Internal Union

3.4.5 Union

The algorithm for the union of relations R_1 and R_2, assuming as usual the requirement for sequential output of the result relation, has two stages. First, each tuple of R_1 is output in succession, and each one which also occurs in R_2 is associatively marked to avoid duplication in the result relation. Second, all unmarked R_2 tuples are output. The procedure is detailed in Algorithm 3.5. From the execution counts, it may be seen that the algorithm requires time

$$(r + c(R_1) + 2)T_p$$

It should be noted that, as in the case of the join operator, this algorithm is asymmetric with respect to R_1 and R_2, and is more efficient when R_1 is chosen to be the smaller of the two argument relations. The techniques discussed in Subsection 3.4.2 may thus be employed to optimize the efficiency of the evaluation of union on the basis of the relative sizes of its argument relations.

3.4.6 Intersect

In Algorithm 3.6, which computes the intersection of relations R_1 and R_2, each tuple of R_1 is examined in turn, and an associative probe is performed to determine whether the tuple in question is also a member of R_2. It is easily seen that

$$(2c(R_1) + 1)T_p$$

is required to intersect two relations in PAM. Again, the dependence of our result on the choice of R_1 should be noted. Selection of the smaller argument relation for R_1 is in fact somewhat more important in the case of set intersection than set union because of the larger relative contribution of the cardinality of R_1 to total execution time.

It is interesting to compare our algorithm for set intersection with the one presented for the join operator. Note that set intersection may be regarded as a special case of natural join in which the compound join attributes are exactly the set of *all* attributes of the argument relations. In the case of intersection, though, we know that no two tuples in an argument relation can have the same value for this compound join attribute, since relations are in fact sets, and are thus prohibited from containing duplicate tuples as elements. This is precisely the sort of "additional information" discussed earlier which must, in the case of the general join, be determined by reference to the semantics of the particular database at hand. Because this information is available on purely *structural* grounds in the case of intersection, our intersect algorithm avoids the probe that is always necessary to detect exhaustion of all R_1 tuples having the current join attribute value. In the case of those join attribute values that match some R_2 tuple, an additional probe is saved over the case of the general


```
procedure intersect(R1, R2),
    begin
    for each t1 of R1
      with not flag
      set flag and do              (c(R1) + 1 probes)
          if exists t2 in R2       (c(R1) probes)
            with t2 = t1 do
                output t1;
```

Algorithm 3.6. Internal Intersect

join, for much the same reason.

Recent work by Trabb Pardo [1978] on the complexity of set intersection on a von Neumann machine suggests another interesting perspective on our associative algorithm for intersection. Trabb Pardo considered two strategies for representing and intersecting sets of unstructured elements (as distinguished from tuples having internal attribute-value structure, as in the relational algebra). The first involves the representation of sets as *tries*, which are intersected through a process of parallel traversal. The second approach, which permits extremely fast intersections, is closely related to our own algorithm, but uses hashing functions to approximate the process of associative retrieval on a von Neumann machine. Like our algorithm, Trabb Pardo's hashed intersection algorithm searches for the presence of each R_1 tuple, in turn, within R_2, and intersects in time linearly proportional to the smaller argument relation.

In the case of intersection (as opposed to the more general join operator), Trabb Pardo's "pseudo-associative" intersection algorithm in fact appears to offer comparable performance to the associative scheme described here. It is in the more general case of the natural join, where result tuples may be generated based on a *partial* match between the corresponding attributes of the argument relations, that the argument for a non-von Neumann architecture is strongest.

Extending the use of hashed search to the case of the general natural join in the most obvious way would require that each tuple be hashed in more than one way to provide for natural joins over different compound attributes. Since the set of compound attributes on which a join might be based is equivalent to the powerset over the simple attributes, the number of such hashings is in fact exponential in the number of simple attributes. At the cost of a non-standard, but economically feasible, hardware design, the architecture and algorithms we have described permit the straightforward and efficient generalization of the associative approach to set intersection to the more general case of the relational join.


```
procedure setDifference(R1, R2),
    begin
    for each t1 of R1
      with not flag
      set flag and do                    (c(R1) + 1 probes)
          if not exists t2 in R2         (c(R1) probes)
            with t2 = t1 do
                output t1,
```

Algorithm 3.7. Internal Set Difference

3.4.7 Set difference

The algorithm for set difference (Algorithm 3.7), where R_1 is the set minuend and R_2 is the set subtrahend, is quite similar to that for intersection. As in the case of intersection, evaluation of the set difference operator requires time

$$(2c(R_1) + 1)T_p \quad ,$$

but does not offer the freedom to choose R_1 for maximum efficiency.

3.5 External Evaluation

In this section, we will describe the algorithms for evaluating the relational algebraic primitives in the case where the argument relation(s) exceed the capacity of PAM. The seven relational operators may be divided into three categories according to the general manner in which they are externally evaluated. The first category includes the two unary operators select and restrict, whose external evaluation algorithms are the least complex (both in the sense of perspicacity and efficiency) of the seven. The second category contains the single remaining unary operator, project, whose external evaluation is made more complex by the need to avoid duplicate result tuples. The final category is comprised of the four binary operators, equi-join, union, intersection and set difference, whose tuples are set into correspondence using a generalization of the category two algorithm.

The algorithms for evaluation of the operators in the second and third categories are each based on the *partitioning* of the argument relation (or in the case of category three, relations) into *disjoint buckets* (or disjoint *shared* buckets, in category three). Typically, one such bucket (which, in the case of the category two operators, will in general include tuples from both argument relations) is transferred into PAM during each successive revolution of SAM, and the corresponding internal operation performed. In each case, partitioning is accomplished by associatively examining the values of some (compound) *key* attribute in the argument relation(s), defined as follows for each of the category two and three algorithms. In the case of projection, the key is the (compound) projected attribute of the single argument relation. For an external join, the (compound) join attribute in each of the two argument relations are defined as the keys. In the case of the three conventional set operators (union, intersection

and set difference), *all* attributes in the argument relations are included in the key.

In this section, we will describe and analyze the algorithms for transferring successive segments of large argument relations from SAM into PAM in the case of the operators belonging to the first, second and third external evaluation categories, respectively.

3.5.1 Select and Restrict

Selection and restriction differ from the other relational algebraic operations in that they can be evaluated using only the per-track logic of the SAM device, and hence do not require that successive segments of their argument relation be read into PAM for internal evaluation. Very little need be said about the external evaluation of the select operator, since the retrieval of all tuples of a given relation having values from selected attributes which match explicitly specified constants is in fact the central primitive operation characterizing a SAM device. As in the case of CASSM, RAP and RARES, our architecture thus performs external selection in constant time, independent of the size of the argument relation, assuming only that the argument relation is no larger than the capacity of the secondary associative storage device, and that the size of the result relation is does not exceed the bandwidth and buffering limitations of the system. Under these assumptions, a single selection requires time T_s, the time for one revolution of SAM. Indeed, our assumptions raise a number of interesting practical questions that must be considered by the designer of a practical system; these issues have been raised by other database machine researchers, however, and will not be given further attention in the current dissertation.

As noted in Section 3.2, the our specifications for the SAM device also permit the restriction operator to be performed entirely within the SAM device, since the per-track logic is itself capable of testing for equality among the attributes of a single tuple. In contrast with the case of internal evaluation, external restriction thus has the same complexity as external selection, requiring time T_s under the assumptions specified above.

3.5.2 Project

As we have noted earlier in this dissertation, it is the problem of redundant tuple elimination that makes projection a substantial computational task in most applications. In the case where the argument relation is no larger than the capacity of PAM, redundant tuples are implicitly eliminated in the course of the internal projection algorithm. In order to extend the projection algorithm to the problem of external evaluation, however, we must first partition the large argument relation into a set of *key-disjoint buckets*. *Buckets* are defined as non-intersecting subsets of tuples from a given relation, a set of buckets is called *key-disjoint* if no bucket contains any tuple whose key - which in the case of projection is the value of the projected compound attribute—is the same as that of some tuple belonging to a different bucket.

In most cases, the partitioning and transfer algorithms described in Section 3.6 will tend to produce buckets no larger than the capacity of PAM. Given such a partitioning of the argument relation, external projection is effected by reading each bucket into PAM in succession and using the fast associative capabilities of PAM to project the tuples over the key. In the case where a bucket exceeds the capacity of PAM, the procedure is complicated somewhat, although the aggregate effect of such "PAM overflows" on the efficiency of the external evaluation algorithm will be negligible under most conditions.

To illustrate the notion of key disjoint buckets, let us consider a projection over the second attribute of the following binary, integer valued relation, which we will assume to be stored on SAM.

2	7
3	1
4	7
8	7
9	3
3	2
4	1
2	3

Extracting the second attribute without removing duplications yields two instances of the value 1, one of the value 2, two of the value 3 and three of the value 7. Supposing (unrealistically, of course) that PAM has a capacity of five such two attribute tuples, we might bring all tuples having a key value of either 1 or 7 into PAM during a single cycle for internal projection. It is significant that the values represented in a given PAM load need not be contiguous; indeed, the values 1 and 7 are non-contiguous within the projected domain of our example. It is required only that if *any* tuples having the key 1 are brought into PAM on some given cycle, then -- in the absence of PAM overflow -- *all* such tuples are in fact collected on the same cycle.

Let us now consider the modifications necessary to this algorithm in order to accomodate any instances of PAM overflows, occuring when a single bucket exceeds the capacity of PAM. The simplest case -- and by far the most common statistically, as we shall see in Subsection 3.6.2 -- is that of a partition which exceeds the size of PAM by less than 50%, and can thus be divided into three *sub-buckets* A, B and C, any two of which can fit into PAM at a given time. During one SAM revolution, sub-buckets A and B are transferred into PAM and projected over the attribute in question. During the next SAM revolution, the tuples of sub-bucket B are replaced in PAM by those of sub-bucket C, and

following another internal evaluation phase, those of sub-bucket A are replaced by those of sub-bucket B. In this manner, all possible pairs of sub-buckets, and hence, all possible pairs of tuples, are submitted to internal projection in PAM at some point. Generalizing this procedure to the case where x tuples are assigned to a given bucket ($x > a$), a total of

$$\frac{n(n-1)}{2}$$

SAM revolutions are found to be required, where

$$n = \left\lceil \frac{2x}{a} \right\rceil \quad (a < x)$$

In the worst case (corresponding to the situation where *all* key values fall within a given segment, and must thus be assigned to the same partition), external projection thus has a complexity of $O(n^2)$ (albeit with very small constants). In most cases, however, the partitioning and transfer algorithms which we will consider should insure that the effects of PAM overflow are dominated by the lower-complexity terms.

It should be clear that the operation of partitioning the argument relation into key-disjoint buckets on the basis of matching values of the compound key attribute is at the heart of the process of efficient external projection. Because a similar partitioning process, based on the key attributes of *both* argument relations, is involved in the external evaluation of the equi-join, union, intersection and set difference operators, we have chosen to consider the details of partitioning as a separate topic in Section 3.6.

3.5.3 Join, union, intersect and set difference

As is the case for projection, external evaluation of the equi-join, union, intersection and set difference operators cannot be performed efficiently within the SAM device alone. Again, it is necessary to transfer successive portions of the argument relations into PAM for internal evaluation on the basis of associatively identified characteristics of the key attributes. In the case of category three evaluation, though, each bucket is in general comprised of tuples from *both* of the two argument relations whose keys satisfy the current criteria for that bucket. The two argument relations are thus partitioned into what we shall call *key-disjoint shared buckets*, which may be regarded as a variant of the notion of key disjoint buckets introduced in the previous subsection.

Specifically, a *shared bucket* is defined as a set of tuples from either or both of the argument relations R_1 and R_2. Again, the partitioning and transfer algorithms described in Section 3.6 insure that the size of the great majority of such buckets (including both R_1 and R_2 tuples) will not exceed the capacity of PAM. A set of shared buckets is called key-disjoint if no bucket contains any tuple whose key is the same as that of some tuple belonging to a different bucket. It should be recalled that the key of an equi-join is the (possibly compound) join attribute, while in the case of the unstructured set operators, the key is comprised of all attributes taken together. In the latter case, the key-disjointness condition thus reduces to the requirement that no bucket contain a tuple of one argument relation which is also present within some other bucket (necessarily as part of the other argument relation).

As an example, consider the case of an equi-join of the following two integer valued relations R_1 (appearing on the left) and R_2 (on the right) over the second attribute of R_1 and the first attribute of R_2:

2	7
3	1
4	7
8	7
9	3
3	2
4	1
2	3

7	8
2	5
6	3
2	6
1	5

Assuming again a PAM capacity of 5 binary tuples, one possible partitioning would assign to the first bucket all R_1 tuples whose second attribute has either 1 or 3 as its value and all R_2 tuples whose first attribute has either 1 or 3 as its value—specifically, the R_1 tuples (3 1), (9 3), (4 1) and (2 3), together with the R_2 tuple (1 5). (It is perhaps worth mentioning at this point that the identity of the relation to which each such tuple belongs must be included as part of its representation within PAM.) The second bucket might contain all tuples having 7 as the value of the join attribute, with a final bucket for keys of value 2 or 6. Again, it should be noted that the key values included within a given bucket need not fall within a single contiguous range. Indeed, the efficiency of one of the two partitioning algorithms described in the following section is dependent on the admissability of non-contiguously defined buckets.

The procedure for *recovery* from PAM overflows in the course of external joining (invoked in the event that the *combined* size of the two argument relations exceeds the capacity of PAM) is somewhat different from that employed in external projection. The algorithm divides both R_1 and R_2 into sub-buckets, each no larger than half the capacity of PAM, each pair of sub-buckets, one chosen from R_1 and one from R_2, is then transferred into PAM in succession

If x_1 tuples from R_1 and x_2 tuples from R_2 are assigned to the bucket in question ($x_1 \mid x_2 > a$), this recovery procedure requires exactly $n_1 n_2$ SAM revolutions, where

$$n_1 = \left\lceil \frac{2x}{a(R_1)} \right\rceil$$

and

$$n_2 = \left\lceil \frac{2x}{a(R_2)} \right\rceil$$

3.6 Partitioning and Transfer

Since the external evaluation of each of the relational algebraic operators with the exception of selection and restriction is dependent on the partitioning of the argument relation into key-disjoint (possibly shared) buckets, we now turn our attention to the manner in which this process may be efficiently executed. We will consider two techniques for partitioning large argument relations into key-disjoint buckets. The two schemes, which we call the *domain histogram* and *hashing* methods, impose somewhat different requirements on the logic and memory that must be associated with each functional head, and differ slightly in efficiency. Independent of its merits as a practical algorithm for incorporation in an actual system, the domain histogram method is of interest by virtue of its relationship to previous work on associative sorting techniques. The process of domain histogram partitioning will be considered in this context in Subsection 3.6.1. When supported by the available per-track hardware, however, the hash partitioning scheme, described in Subsection 3.6.2, should generally be somewhat faster, and is also more amenable to statistical analysis.

3.6.1 Domain histogram partitioning

The domain histogram method is closely related to a technique introduced by Lin [1977] for sorting external files stored on an associative head-per-track disk device such as SAM. Lin's *bucket sort* algorithm assumes, as does our scheme, that the file can be stored entirely on the associative disk device, and thus that each data entity passes under an intelligent processing unit exactly once per revolution. The scheme functions in a manner analogous to that we have described for external evaluation of the relational algebraic operators, reading one bucket from the external file into a primary random access memory during each successive revolution of the disk. By contrast with the relational operators, however, the task of sorting requires that each partition be comprised of tuples whose sort domain—the compound attribute whose value is to determine the sorted order—contains contiguous values. Note that if the sort domain values were known a *priori* to be uniformly distributed over some range $[x_{min}, x_{max}]$, the file could be divided into $h = (c/a)$ buckets, each containing a tuples of relation R (ignoring a few boundary conditions), with the i-th inter-bucket boundary being

$$\frac{x_{min} + i(x_{max} - x_{min})}{h}$$

Each such bucket would correspond to one contiguous range of sort domain values, so that successive buckets could be read into primary storage in a monotonic sequence of their sort key ranges, then internally sorted, and the resulting file output in fully-sorted order.

Unfortunately, most files of practical interest deviate substantially from this assumption of uniform sort domain distribution. Lin's solution involves dividing the domain into a large number of equal sized intervals whose size is

small by comparison with P. During a single preliminary revolution of the associative disk device, a count is taken of the number of tuples of R whose sort domain values fall within the bounds of each of these smaller intervals, forming what is called a *domain histogram*. The lowest-valued k intervals are then combined to form the first bucket, with k chosen as large as possible such that the resulting bucket would fit within available primary storage (based on the counts of each such interval and the fixed tuple size). This first bucket is then tranferred into primary storage for internal sorting. On each successive revolution of the associative disk device, another such bucket is identified in a similar manner and read into primary storage.

As an example, consider the case of a file of 10-byte tuples whose integer-valued sort domain is bounded by the values 0 and 99. We might first divide the domain into ten equal intervals, and obtain the following counts for the number of tuples whose sort domain values fall within each interval:

$[x_1, x_2]$:	*count*
[0, 9]:	53
[10, 19]:	81
[20, 29]:	27
[30, 39]:	59
[40, 49]:	2
[50, 59]:	14
[60, 69]:	11
[70, 79]:	28
[80, 89]:	36
[90, 99]:	91

Assuming 2000 bytes of available storage, the first three intervals, together

occupying (53 | 81 | 27) 10 = 1610 bytes, would constitute the first bucket. Thus on the first revolution (following the one required for histogram creation), all tuples whose sort domain values fell between 0 and 29 would be read into primary storage in the order they were encountered on the disk. The algorithm for internal projection would then be applied to this first bucket of tuples, and the result output. On the next revolution, all tuples in the five intervals bounded by (30, 89) would be read and processed internally; the third bucket would consist of all tuples within the bounds of the final interval.

The associative bucket sort algorithm can be applied to the problem of key-disjoint partitioning and transfer by using the key of the argument relation (in the case of projection) or relations (in the case of join, union, intersection and set difference) in the same way as the sort domain is used in the bucket sort algorithm. As it happens, though, the loosening of the contiguity constraint in favor of the weaker requirements of key-disjoint partitioning makes possible a modest refinement of this technique when applied to the relational algebraic operators. Note that if the interval bounded by [80, 89] is added to the first partition (which, unlike the interval immediately following the first partition, would not result in a PAM overflow), and the entire third partition then merged with the second (which would now have enough room), only two SAM revolutions (plus the one for histogram construction) would be required to pass the relation through PAM. The contiguity requirement thus makes it necessary to expend one extra SAM revolution by comparison with a different assignment of intervals to buckets which would be possible in the absence of this requirement. Indeed, Lin has observed that the average bucket size obtained using the bucket sort algorithm may be as small as half the capacity of the available primary store in a worst case situation, resulting in up to twice the optimal number of disk revolutions.

The task of finding an optimum partitioning of the relation from the viewpoint of minimizing the required number of SAM revolutions is an example of a *bin packing* problem, whose exact solution is unfortunately NP-complete. In practice, however, one of several known linear time heuristic algorithms for non-optimal, but typically reasonably effective, bin packing can be used to improve the performance of the partitioning of relations using domain histograms. (As these algorithms seem to constitute a separable and fairly well reported area of work, they will not be given further attention in this dissertation.)

Having identified a group of interval sets that do a reasonable job of controlling the number of buckets, all tuples whose key falls within the set of interval ranges defining a particular bucket must be retrieved during a single revolution of SAM. This imposes stronger requirements on the capabilities of the per-track logic than those required by the unoptimized algorithm, since more than one range specification must be checked for each tuple passing under the head. Although there are several possible ways in which this operation might be performed, most depend on a fixed limit on the number of non-contiguous ranges used to define each bucket, thus constraining the bin packing problem in an interesting way.

Three factors are worth mentioning with regard to the choice of interval size. First, the expected amount of wasted PAM space after bin packing (manifested in a larger number of buckets, and hence, additional SAM revolutions) is directly related to the size of the intervals. Second, the likelihood that those tuples whose keys fall within a *single* interval will exceed the capacity of PAM (thus causing a PAM overflow regardless of the chosen partitioning) varies inversely with interval size. Note, however, that there is no interval size small enough to guarantee that no overflow will occur; the recovery pro-

cedures outlined in Section 3 5 are necessary to provide for the occurence of PAM overflow, however unlikely.

Finally, we note that the choice of an extremely small interval size is not without substantial cost, as each per-track logical unit would almost certainly (at least within the context of the designs we have considered) require a quantity of random access memory bearing an inverse linear relationship to interval size. To see why this is the case, note that the total number of interval count increments required during the first (histogram creation) revolution of SAM is exactly c (assuming, for simplicity, a single argument relation). The bandwidth necessary to perform all of these increments directly on one single-ported random access memory could easily be several orders of magnitude too great in a typical practical application. All of the solutions we have considered seem to be essentially equivalent to the provision of a number of random access words within each per-track unit which is equal to the maximum number of intervals into which the domain can be divided. The individual subtotals from each per-track logical unit may then be summed to obtain the final counts for each interval. (Although the time required for this final summation is proportional to the number of SAM heads in the absence of n-argument adding hardware, this delay, which occurs only once per operator evaluation, should ordinarily be insignificant by comparison with the cost of associative retrieval.)

In attempting to rigorously evaluate the average case behavior of the domain histogram method, we are faced with the need to make fairly strong (and problematic, given our limited current understanding of the actual and potential use of such systems) assumptions about the incidence of PAM overflow. In the case of the hash partitioning method, on the other hand, a much weaker set of assumptions yields an analytically tractable model for use in computing the average case cost -- which in fact turns out to be linear and small -- of PAM

overflows. Since the hash partitioning technique is probably superior in most applications to the method currently under discussion (at least under the assumption of suitable per-track logical capabilities), we will thus omit a detailed average case analysis of the overflow scheme as applied to domain histogram partitioning, but include such a treatment in our analysis of the hash-based scheme.

3.6.2 Hash partitioning

Let us now turn our attention to the hash-based scheme for partitioning and transferring the argument relation. The intent of this algorithm is to manage the problem of non uniform distribution of the key by assigning tuples to PAM-sized buckets using a hashing function. The algorithm requires that each per-track unit be capable of sequentially computing a hashing function on the compound attribute in question, and of outputting all tuples for which the resulting hashed value falls within a specified range. Because the algorithm does not require the ability for a dynamic choice of the *range* of the hash function, the requirement for real-time hashing is well within the capabilities of the sort of simple and inexpensive hardware that would be required in a practical per-track logical unit. One implementation, for example, would combine the entire compound attribute into a single, fixed length "signature word" (of, say, 16 bits), by computing the *exclusive or* of each two byte segment with the current accumulated signature word as it passes under the head. In the discussion which follows, we assume that the hashing function maps all keys onto a range $[0, H_{max}]$.

In the interest of simplicity, we will first consider the case of a single relational argument. In the first step of the algorithm for category two hash-based partitioning, the range of the hash function is divided into h equal *hash intervals*, where

$$h = \left\lceil \frac{(1 + W)c}{a} \right\rceil .$$

W (for "waste factor") is a fixed system parameter, ordinarily much smaller than one. The number of hash intervals is thus chosen to be somewhat larger than the size of the relation in "PAM-fulls". (We assume that the size in bytes of each stored relation is immediately available or easily determinable, so that

this operation requires negligible time.) During each SAM revolution, all tuples whose keys hash to a value within a single hash interval are transferred into PAM, providing their combined size does not exceed the capacity of PAM.

In the absence of overflows, exactly h SAM revolutions, requiring time hT_r, are necessary to transfer all buckets of the argument relation(s) into PAM. Whenever x, the number of tuples assigned to the current bucket, is greater than (c/h) by a factor of more than W, however, an overflow occurs, resulting in the expenditure of more than one SAM revolution for the bucket in question; the exact number of revolutions depends on the ratio of x to (c/h). In the general case where an average of v extra "overflow revolutions" are required per bucket, the time required is exactly

$$(1 + v)hT_r$$

The central concern of our analysis is the derivation of an upper bound on the average case value of v.

By comparison with the domain histogram algorithm, the randomizing property of the hashing scheme permits a relatively accurate statistical evaluation of the number and extent of PAM overflows to be expected in the course of hash partitioning without excessively stringent assumptions regarding the distribution of the key values. (Our analysis is dependent, of course, on the assumption that the distribution of hash values, given a large set of keys, will be close to uniform over the range $[1, H_{max}]$; this may not in fact always be the case.) The analysis is based on the treatment of the partitioning process as a set of c independent Bernoulli trials, one for each tuple in the relation, with each trial defined as successful if the tuple in question falls within the current hash interval, and as unsuccessful otherwise. The number of tuples

which will be assigned to any given bucket is thus a binomially distributed random variable whose probability of being equal to some particular value k is exactly

$$\binom{c}{k}\left(\frac{1}{h}\right)^{k}\left(1-\frac{1}{h}\right)^{(c-k)} .$$

Unless there is a very small number of tuples per PAM load, this function is well approximated by the Gaussian distribution

$$\phi\left(\frac{x-\eta}{\sigma}\right)=\frac{1}{\sqrt{2\pi}\,\sigma}e^{-\frac{(x-\eta)^2}{2\sigma^2}}$$

having mean

$$\eta=\frac{c}{h}$$

and variance

$$\sigma^2=\left(1-\frac{1}{h}\right)\eta .$$

Furthermore, both η and σ^2 approach

$$\frac{a}{1+W}$$

as c grows large, and are thus asymptotically independent of the size of the argument relations

Note that this approximation differs from that most commonly employed in analyzing hash coding behavior in database management applications (see **Wiederhold** [1977], for example) In the more common use of hashing, the **expected** value of x is typically quite small, so that the corresponding function is better approximated by a Poisson distribution When the η is reasonably large, however, a normal distribution provides a better approximation As a practical rule of thumb, the Gaussian approximation, which is justified in the

limit by the DeMoivre Laplace theorem, is very good whenever the quantity

$$\left(\frac{1}{p}-\frac{1}{S}\right)^{t}$$

is less than about 0 1, which should be true in most conceivable practical cases

The expected number of overflow revolutions may thus be estimated by

$$v=\sum_{i=2}^{\infty} i\left(1+\frac{1}{2}\right)\int_{(i+1)a/2}^{(i+1)a/2}\phi\left(\frac{x-\eta}{\sigma}\right)dx .$$

For purposes of obtaining a simple upper bound, the discrete summation may be eliminated by substituting $2x/a$ for i within each term, so that a constant expression equal to the lower limit of integration is replaced by the variable of integration within that range, which must necessarily be larger. This yields

$$v<\int_{a}^{\infty}\frac{x}{a}\left(\frac{x}{a}+\frac{1}{2}\right)\phi\left(\frac{x-\eta}{\sigma}\right)dx$$

$$=\left(\frac{1}{2a}\right)^{2}\left\{(2\sigma^2+\eta(2\eta+a))\left(1-\Phi\left(\frac{a-\eta}{\sqrt{2}\,\sigma}\right)\right)+(2\eta+3a)\phi\left(\frac{a-\eta}{\sqrt{2}\,\sigma}\right)\right\} ,$$

where

$$\Phi(x)=\int_{-\infty}^{x}\phi(y)\,dy ,$$

which has no closed form solution, but whose values for specific x are available in tabular form (see, for example, Hoel, Port and Stone [1971])

v is thus independent of the size of the argument relations, and since h varies linearly with argument size, the time

$$(1+v)hT_{s}$$

for partitioning and transfer is of linear complexity in the size of the argument relations (Since the algorithm for internal projection is also linear, the corresponding external algorithms are linear) The time required is, however, inversely related to W, the waste factor, and directly related to a, the capacity in tuples of PAM Calculations using a range of typical c, t, P and h values suggest that a very modest W (say, on the order of 0 1) should generally suffice to make the cost of overflow recovery negligible by comparison with the complexity component due to the transfer of non overflowing buckets

The algorithm for hash based external evaluation of the join, union, intersect and set difference operators is analogous to the one described for external projection In the case of the category two operators, the number of *hash intervals*, h, is set equal to

$$h \quad \left\lceil (1 + W)\left(\frac{c(R_1)}{a(R_1)} + \frac{c(R_2)}{a(R_2)}\right)\right\rceil$$

Analysis of the average case time complexity of the category two operators is similar to that presented above for projection, the primary differences being due to substitution of $n_1 n_2$ for $n(n-1)/2$ as the number of SAM revolutions required for recovery from PAM overflow As in the case of projection, such overflows make only a linear contribution to the cost of category two evaluation

In practice, the time required for evaluation of both the category one and category two operators should ordinarily be quite close to the sum of

1 the time required for a number of SAM revolutions equal to the size of the argument relation (or in the case of category two, the combined size of the two argument relations) in "PAM-fulls", and

2 the time required for internal evaluation of the operator in question

In the case where the argument relation(s) are large, this may represent a very substantial improvement on the results attainable using a database

machine based on an associative secondary storage device alone, as in the RAP, CASSM and RARES designs

3.7 Summary

In this chapter, we have proposed a non-von Neumann machine architecture for the efficient large scale evaluation of relational algebraic database primitives. The design is based on a content-addressable primary storage unit called PAM and a rotating logic-per-track associative device called SAM, both based on existing, and in the near future, economically feasible, technology. The machine we have described functions in much the same way as several proposed and already implemented database machines for the operations of selection and restriction, but appears to offer a significant performance advantage in the case of project, join, and the unstructured set operators.

Specifically, the time required for external selection and restriction is independent of the size of the argument relation, being equal to the time for one revolution of SAM under the assumptions enumerated in the chapter. This result substantially improves upon the best known general algorithms for evaluating these operations on a von Neumann machine, but is essentially equivalent to those obtained on most of the database machines reviewed in Subsection 3.1.2. The time required for external projection, join, union, intersection and set difference, on the other hand, is roughly that required for a number of SAM revolutions equal to the combined size of the argument relations in "PAM-fulls" plus the (also linear) time required for internal evaluation of the operator in question. This latter result represents an $O(\log n)$ improvement over the best presently known methods on a von Neumann machine, and appears to offer a large linear factor improvement (roughly proportional to the capacity of PAM) over the best reported results involving a specialized database machine architecture having comparable hardware complexity.

It must be acknowledged, however, that we have left many details unspecified, have made a number of assumptions deserving careful examination, and have not yet performed the sorts of detailed comparisons that would justify a confident claim that the architecture we have described is in fact more suitable for practical application than those already proposed in the literature. It is hoped that the readers of this dissertation will contribute to the process of critical review necessary to adequately assess the merit of the approach we have suggested.

APPENDIX

AXIOMS DEFINING THE MATCH SEMANTICS

The following 26 axioms make up the match specification which defines the semantics of matching for our knowledge based description language. In order to insure faithful reproduction of the axioms as they are actually used in our working demonstration system, we have chosen to present them in their original infix LISP syntax instead of the symbolic prefix form used for pedagogical purposes in the text, which is admittedly probably somewhat easier to read. For the most part, the interpretation of the constituent logical formulæ should be quite clear. Several details of the LISP syntax of our well formed formulæ should be noted, however.

First, conjunctive ("And") and disjunctive ("Or") formulæ may each have any number of arguments. Second, either a single well-formed formula or a list of (implicitly conjoined) formulæ may appear as the body of a defined predicate or of an existentially or universally quantified formula. (This is merely a notational convention, intended to obviate the need for an explicit "And" surrounding each such body.)

Specifically, defined predicates have the following form in LISP syntax:

(*(defined predicate name)* (*formal parameters*) (*body*)) ,

where (*formal parameters*) is a (LISP) list of identifiers and (*body*) is a list of one or more well-formed formulæ.

The syntax of existentially quantified formulæ is as follows:

(Exists (∃ *variables*) (*body*)) ,

where (∃ *variables*) is a list of identifiers.

The restricted form of universally quantified formula supported by the LSEC algorithm has the syntax

(For-all (∀ *variables*) (*qualification clause*) (*body*)) .

where (V *variables*) is again a list of identifiers. The formula

(For-all (x) (Q x) (B x)) ,

for example, is interpreted as

$$\forall x\ Q(x) \supset B(x)$$


```
(Dtion-imp-dtion
  (tar-dtion pat-dtion)
  (Or
    (Dtion-directly-imp-dtion tar-dtion pat-dtion)
    (Exists
      (antecedent-dtion consequent-dtion)
      (Dtion-directly-imp dtion consequent-dtion pat-dtion)
      ('Antecedent-Consequent' antecedent-dtion consequent-dtion)
      (Dtion-imp-dtion tar-dtion antecedent-dtion))))
```

```
(Dtion-directly-imp-dtion
  (tar-dtion pat-dtion)
  (For-all
    (pat-dtor)
    ('Dtion_Dtor' pat-dtion pat-dtor)
    (Dtion-imp-dtor tar-dtion pat-dtor)))
```

```
(Dtion-imp dtor
  (tar-dtion pat-dtor)
  (Or
    ('Dtor Dtype' pat-dtor 'NULL)
    (And
      ('Dtor Dtype' pat-dtor 'INDIVIDUAL)
      (Dtion-imp-individual-dtor tar-dtion pat-dtor))
    (And
      ('Dtor-Dtype' pat-dtor 'PERSPECTIVE)
      (Dtion-imp-perspective dtor tar-dtion pat-dtor))
    (And
      ('Dtor-Dtype' pat-dtor 'SET-OF)
      (Dtion-imp-set-of-dtor tar-dtion pat-dtor))
    (And
      ('Dtor Dtype' pat-dtor 'SET-WITH-ALL-OF)
      (Dtion-imp-set-with-all-of-dtor tar-dtion pat-dtor))
    (And
      ('Dtor-Dtype' pat-dtor 'SET-WITH-ANY-OF)
      (Dtion-imp set-with-any-of-dtor tar-dtion pat-dtor))
    (And
      ('Dtor Dtype' pat-dtor 'INVOLVES)
      (Dtion-imp-involves-dtor tar-dtion pat-dtor))
    (And
      ('Dtor-Dtype' pat-dtor 'DISJUNCTIVE)
      (Dtion imp disjunctive dtor tar dtion pat dtor))))
```



```
(Dtion-imp-individual-dtor
  (tar-dtion pat-individual-dtor)
  (Exists
    (tar-dtor)
    (Dtor-imp individual-dtor tar dtor pat-individual dtor)
    ('Dtion-Dtor' tar-dtion tar dtor)))
```

```
(Dtor-imp-individual-dtor
  (tar-dtor pat-individual-dtor)
  (Individual-dtor-imp-individual-dtor tar-dtor pat-individual-dtor)
  ('Dtor-Dtype' tar-dtor 'INDIVIDUAL))
```



```
(Individual-dtor-imp-individual dtor
  (tar-individual-dtor pat-individual-dtor)
  (Exists
    (individual)
    ('Individual-dtor -Individual' pat-individual-dtor individual)
    ('Individual-dtor -Individual' tar-individual-dtor individual)))
```

```
(Dtion-imp-perspective-dtor
  (tar-dtion pat-perspective-dtor)
  (Exists
    (tar-dtor)
    (Dtor-imp-perspective-dtor tar-dtor pat-perspective-dtor)
    ('Dtion-Dtor' tar-dtion tar-dtor)))
```



```
(Dtor-imp-perspective-dtor
  (tar-dtor pat-perspective-dtor)
  (Perspective-dtor-imp-perspective-dtor tar-dtor pat-perspective-dtor)
  ('Dtor-Dtype' tar-dtor 'PERSPECTIVE))
```

```
(Perspective-dtor-imp-perspective-dtor
  (tar-perspective-dtor pat-perspective-dtor)
  (Exists
    (prototype)
    ('Perspective 'Prototype' pat-perspective-dtor prototype)
    ('Perspective-Prototype' tar-perspective-dtor prototype))
  (For-all
    (pair-name)
    (Exists
      (pat-pair-filler)
      ('Object-Slot-Filler' pat-perspective-dtor pair-name pat-pair-filler))
    (Exists
      (pat-pair-filler tar-pair-filler)
      ('Object-Slot-Filler' pat-perspective-dtor pair-name pat-pair-filler)
      ('Object-Slot-Filler' tar-perspective-dtor pair-name tar-pair-filler)
      (Dtion-imp dtion tar-pair-filler pat-pair-filler))))
```



```
(Dtion-imp-set-of-dtor
  (tar-dtion pat-set-of-dtor)
  (Exists
    (tar-dtor)
    (Dtor-imp-set-of-dtor tar-dtor pat-set-of-dtor)
    ('Dtion-Dtor' tar-dtion tar-dtor)))
```

```
(Dtor-imp-set-of-dtor
  (tar-dtor pat-set-of-dtor)
  (Or
    (And
      (Set-of-dtor-imp-set-of-dtor tar-dtor pat-set-of-dtor)
      ('Dtor-Dtype' tar-dtor 'SET-OF))
    (And
      (Set-with-exactly-dtor-imp-set-of-dtor tar-dtor pat-set-of-dtor)
      ('Dtor-Dtype' tar-dtor 'SET-WITH-EXACTLY))))
```



```
(Set-of-dtor-imp-set-of-dtor
  (tar-set-of-dtor pat-set-of-dtor)
  (Exists
    (tar-sub-dtion pat-sub-dtion)
    ('Set-of-dtor Sub-dtion' pat set of dtor pat-sub dtion)
    (Dtion-imp-dtion tar-sub-dtion pat-sub-dtion)
    ('Set-of-dtor-Sub-dtion' tar-set-of-dtor tar-sub-dtion)))
```

```
(Set-with-exactly-dtor-imp set-of-dtor
  (tar-set-with-exactly-dtor pat-set-of-dtor)
  (Exists
    (pat-sub-dtion)
    ('Set-of-dtor-Sub-dtion' pat-set-of-dtor pat-sub-dtion)
    (Exists
      (tar-sub-dtion)
      (Dtion-imp dtion tar-sub-dtion pat-sub-dtion)
      ('Set-with-exactly-dtor-Sub-dtion'
         tar-set-with-exactly-dtor
         tar-sub-dtion)
      (For-all
        (tar sub-dtion)
        ('Set-with-exactly-dtor-Sub-dtion'
           tar-set-with exactly-dtor
           tar-sub-dtion)
        (Dtion-imp dtion tar-sub dtion pat-sub-dtion)))))
```



```
(Dtion-imp-set-with-all-of-dtor
  (tar-dtion pat-set-with-all-of-dtor)
  (Exists
    (tar-dtor)
    (Dtor-imp-set-with-all-of-dtor tar-dtor pat-set-with-all-of-dtor)
    ('Dtion-Dtor' tar-dtion tar-dtor)))
```

```
(Dtor-imp-set-with-all-of-dtor
  (tar-dtor pat-set-with-all-of dtor)
  (Or
    (And
      (Set-with-all-of-dtor-imp-set-with-all-of-dtor
        tar-dtor
        pat-set-with-all-of-dtor)
      ('Dtor_Dtype' tar-dtor 'SET-WITH-ALL-OF))
    (And
      (Set-with-exactly-dtor-imp-set-with-all-of-dtor
        tar-dtor
        pat-set-with-all-of-dtor)
      ('Dtor-Dtype' tar-dtor 'SET-WITH-EXACTLY))))
```



```
(Set-with-all-of-dtor-imp-set-with-all-of-dtor
  (tar-set-with-all-of-dtor pat-set-with all of dtor)
  (For-all
    (pat-sub-dtion)
    ('Set-with-all-of-dtor_Sub-dtion' pat-set-with all of-dtor pat-sub-dtion)
    (Exists
      (tar-sub-dtion)
      (Dtion-imp dtion tar-sub dtion pat-sub dtion)
      ('Set-with-all-of-dtor_Sub-dtion' tar-set-with-all-of-dtor tar-sub-dtion))))
```

```
(Set-with exactly-dtor imp-set-with all of-dtor
  (tar-set with exactly-dtor pat set-with-all-of-dtor)
  (For-all
    (pat sub-dtion)
    ('Set-with-all-of dtor Sub-dtion' pat-set-with-all-of-dtor pat-sub-dtion)
    (Exists
      (tar-sub-dtion)
      (Dtion-imp dtion tar-sub-dtion pat sub-dtion)
      ('Set-with exactly-dtor-Sub dtion'
         tar set with exactly-dtor
         tar-sub-dtion))))
```



```
(Dtion-imp-set-with-any-of-dtor
  (tar-dtion pat-set-with-any-of-dtor)
  (Exists
    (tar-dtor)
    (Dtor-imp set-with-any-of-dtor tar dtor pat-set with-any-of-dtor)
    ('Dtion- Dtor' tar-dtion tar-dtor)))
```

```
(Dtor-imp-set-with-any-of-dtor
  (tar-dtor pat-set-with-any-of-dtor)
  (Or
    (And
      (Set-with-all-of-dtor-imp-set-with-any-of-dtor
        tar-dtor
        pat-set-with-any-of-dtor)
      ('Dtor-Dtype' tar-dtor 'SET-WITH-ALL-OF))
    (And
      (Set-with-exactly-dtor-imp-set-with-any-of-dtor
        tar-dtor
        pat-set-with-any-of-dtor)
      ('Dtor-Dtype' tar-dtor 'SET-WITH-EXACTLY))
    (And
      (Set-of-dtor-imp-set-with-any-of-dtor
        tar-dtor
        pat-set-with-any-of-dtor)
      ('Dtor-Dtype' tar-dtor 'SET-OF))))
```



```
(Set-with-all-of-dtor-imp-set-with-any-of-dtor
  (tar-set-with-all-of-dtor pat-set-with-any-of-dtor)
  (Exists
    (pat-sub-dtion tar-sub-dtion)
    ('Set-with-any-of-dtor-Sub-dtion'
      pat-set-with-any-of-dtor
      pat-sub-dtion)
    (Dtion-imp-dtion tar-sub-dtion pat-sub-dtion)
    ('Set-with-all-of-dtor-Sub-dtion'
      tar-set-with-all-of-dtor
      tar-sub-dtion)))
```

```
(Set-with exactly-dtor-imp-set-with any-of-dtor
  (tar-set-with-exactly-dtor pat-set-with-any-of-dtor)
  (Exists
    (pat-sub-dtion tar-sub-dtion)
    ('Set-with-any-of-dtor-Sub-dtion'
       pat-set-with-any-of-dtor
       pat-sub-dtion)
    (Dtion-imp-dtion tar-sub-dtion pat-sub-dtion)
    ('Set-with-exactly-dtor Sub-dtion'
       tar-set-with-exactly-dtor
       tar-sub-dtion)))
```



```
(Set-of-dtor-imp-set-with-any-of-dtor
  (tar-set-of-dtor pat-set-with-any-of-dtor)
  (Exists
    (pat-sub-dtion tar-sub-dtion)
    ('Set-with-any-of-dtor-Sub-dtion'
       pat-set-with-any-of-dtor
       pat-sub-dtion)
    (Dtion imp-dtion tar-sub-dtion pat-sub-dtion)
    ('Set-of-dtor Sub dtion'
       tar-set-of-dtor
       tar-sub-dtion)))
```

```
(Dtion-imp-involves-dtor
  (tar-dtion pat-involves-dtor)
  (Exists
    (pat-involved-dtion)
    ('Involves-dtor-Sub-dtion' pat-involves-dtor pat-involved-dtion)
    (Or
      (Dtion-imp dtion tar-dtion pat-involved-dtion)
      (Exists
        (tar-sub-dtion)
        (Dtion-imp-dtion tar-sub-dtion pat-involved-dtion)
        (Dtion-proper-super-dtion tar-sub-dtion tar-dtion)))))
```



```
(Dtion-proper-super-dtion
  (dtion proper-super-dtion)
  (Exists
    (immediate-super-dtor immediate-super-dtion)
    (Dtion-immediate-super-dtor dtion immediate super-dtor)
    (Or
      ('Dtion-Dtor' proper-super-dtion immediate-super-dtor)
      (And
        ('Dtion-Dtor' immediate-super-dtion immediate-super-dtor)
        (Dtion-proper-super-dtion
          immediate-super-dtion
          proper-super-dtion)))))
```

```
(Dtion immediate-super-dtor
  (dtion immediate-super-dtor)
  (Or
    (And
      (Exists
        (slot)
        ('Object-Slot-Filler' immediate-super-dtor slot dtion))
      ('Dtor-Dtype' immediate-super-dtor 'PERSPECTIVE))
    (And
      ('Set-of-dtor-Sub-dtion' immediate-super-dtor dtion)
      ('Dtor-Dtype' immediate-super-dtor 'SET-OF))
    (And
      ('Set-with-all-of-dtor-Sub-dtion' immediate-super-dtor dtion)
      ('Dtor-Dtype' immediate-super-dtor 'SET-WITH-ALL-OF))
    (And
      ('Set-with-all-of-dtor-Sub-dtion' immediate-super-dtor dtion)
      ('Dtor-Dtype' immediate-super-dtor 'SET-WITH-ANY-OF))
    (And
      ('Involves-dtor-Sub-dtion' immediate-super-dtor dtion)
      ('Dtor-Dtype' immediate-super-dtor 'INVOLVES))
    (And
      ('Disjunctive-dtor-Sub-dtion' immediate-super-dtor dtion)
      ('Dtor-Dtype' immediate-super-dtor 'DISJUNCTIVE))))
```



```
(Dtion-imp-disjunctive-dtor
  (tar-dtion pat-disjunctive-dtor)
  (Exists
    (pat-disjoined-sub-dtion)
    ('Disjunctive-dtor-Sub-dtion' pat disjunctive-dtor pat-disjoined-sub-dtion)
    (Dtion-imp-dtion tar dtion pat-disjoined-sub-dtion)))
```

REFERENCES

Ampex Corporation, "9300 Parallel Transfer Disk Drive", product announcement, Redwood City, 1978.

Anderson, Donald R., "Data base processor technology", *Proceedings of the National Computer Conference*, 1976.

Anderson, G. A. and Kain, R. Y., "A content-addressed memory design for data base applications", *Proceedings of the 1976 International Conference on Parallel Processing*, IEEE, New York, pp. 191-195, 1976.

Andrews, D. D., "Interrelationships as a basis for information retrieval", *International Conference on Standards on a Common Language for Machine Searching and Translation*, 1959.

Atherton, P., "Standards for a user-system interface language in online retrieval systems", *Online Review*, vol. 2, no. 1, 1978.

Banerjee, Jayanta, Baum, Richard I., Hsiao, David K. and Kannan, Krishnamurthi, "Concepts and capabilities of a database computer", to appear in *ACM Transactions on Database Systems*, 1979.

Bar-Hillel, Y., "A logician's reaction to recent theorizing on information search systems", *American Documentation*, vol. 8, 1957.

Batcher, K. E., "STARAN parallel processor system hardware", *Proceedings of the AFIPS 1974 National Computer Conference*, vol. 43, AFIPS Press, Montvale, New Jersey, pp. 405-410, 1974.

Baum, Richard I. and Hsiao, David K., "Data base computers—a step towards data utilities", *IEEE Transactions on Computers*, vol. C-25, December, 1976.

Bennerts, R. K., "Development of the Defense Documentation Center Remote On-Line Retrieval System: Past, present and future", Defense Documentation

Center, AD 720 900, Alexandria, Virginia, 1971.

Berra, P. Bruce, "Some problems in associative processor applications to data base management", *Proceedings of the National Computer Conference*, 1974.

Berra, P. Bruce, "Data Base Machines", *ACM SIGIR Forum*, Winter, 1977.

Bobrow, D. G., and Winograd, Terry, "An overview of KRL-0, a knowledge representation language", *Cognitive Science*, vol. 1, no. 1, 1977.

Brandhorst, W. T. and Eckert, P. F., "Document retrieval and dissemination systems", in Cuadra, C. (Ed.) *Annual Review of Information Science and Technology*, Volume 7, American Society for Information Science, Washington, D. C., 1972.

Canady, et al., "A back-end computer for database management", *Communications of the Association for Computing Machinery*, vol. 17, pp. 575-582, October, 1974.

Chang, C. L. and Lee, R. C. T., *Symbolic Logic and Mechanical Theorem Proving*, Computer Science and Applied Mathematics Series, Academic Press, Inc., New York, 1973.

Chu, Y. H., "A destructive-readout associative memory", *IEEE Transactions on Computers*, EC-14, pp. 600-605, August, 1965.

Cleverdon, C. W., "The evaluation of systems used in information retrieval", *International Conference on Scientific Information*, Washington, 1958.

Cleverdon, C. W., "Report on the first stage of an investigation", College of Aeronautics, Cranfield, 1960.

Cleverdon, C. W., "Report on the testing and analysis of an investigation into the comparative efficiency of indexing systems", ASLIB-Cranfield Research

Report, Cranfield, England, 1962.

Codd, E. F., "A relational model of data for large shared data banks", *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, June, 1970.

Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", *Proceedings of the 1971 ACM SIGFIDET Workshop on Data Description, Access and Control*, Association for Computing Machinery, 1971.

Codd, E. F., "Relational completeness of data base sublanguages", in Rustin, Randall (ed.), *Courant Computer Science Symposium 6: Data Base Systems*, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1972.

Cook, K. H., "An experimental on-line system for psychological abstracts", *Proceedings of the American Society for Information Science*, vol. 7, 1970.

Cook, K. H., Trump, L. H., Atherton, P., and Katzer, J., "Large scale information processing systems", Final report to the Rome Air Development Center, Syracuse University, School of Library Science, Syracuse, New York, 1971.

Copeland, G. P., Lipovski, G. J. and Su, S. Y. W., "The architecture of CASSM: A cellular system for nonnumeric processing", *Proceedings of the First Annual Symposium on Computer Architecture*, 1973.

Costello, J. C., and Wall, E., "Recent Improvements in Techniques for Storing and Retrieving Information", dul'ont de Nemours and Co., Wilmington, 1959.

Couranz, G. R., Gerhardt, M. S., and Young, C. J. "Programmable radar signal processing using the RAP", *Proceedings of the Sagamore Computer Conference on Parallel Processing*, Springer-Verlag, New York, pp. 37-52, 1974.

Crane, H. A. and Githens, J. A., "Bulk processing in distributed logic memory", *IEEE Transactions on Computers*, EC-14, pp. 186-196, April, 1965.

Crofut, W. A. and Sottile, M. R., "Design techniques of a delay-line content-addressed memory", *IEEE Transactions on Computers*, pp. 529-534, 1966.

DeFiore, Casper R. and Berra, P. Bruce, "A data management system utilising an associative memory", *Proceedings of the AFIPS National Computer Conference*, vol. 42, 1973.

DeFiore, Casper R., Stillman, C. R., and Berra, P. Bruce, "Associative techniques in the solution of data management problems", *Proceedings of the ACM*, pp. 28-36, 1971.

DeWitt, David J., "DIRECT - A multiprocessor organization for supporting relational database management systems", *IEEE Transactions on Computers*, vol. c-28, no. 6, June, 1979.

Ewing, R. G. and Davies, P. M., "An associative processor", *Proceedings of the AFIPS 1964 Fall Joint Computer Conference*, Spartan Books, Inc., Baltimore, Maryland, pp. 147-158, 1964.

Fife, D. W., Rankin, K., Fong, E., Walker, J. C. and Marron, B. A., "A technical index of interactive information systems", National Bureau of Standards Technical Note 819, 1974.

Finnila, C. A., "The associative linear array processor", *IEEE Transactions on Computers*, February, 1977.

Firschein, O., Summit, R. K. and Mick, C. K., "Use of on-line bibiographic search in public libraries: a retrospective evaluation", *Online Review*, vol. 2, no. 1, 1978.

Flynn, M. J., "Some computer organizations and their effectiveness", *IEEE Transactions on Computers*, pp. 948-960, September, 1972.

Fong, E., "A survey of selected document processing systems", National Bureau of Standards Technical Note 599, 1971.

Foster, Caxton C., *Content Addressable Parallel Processors*, New York, Van Nostrand Reinhold, 1976.

Freeman, R. R., and Atherton, P., "AUDACIOUS - An experiment with an on-line, interactive reference retrieval system using the universal decimal classification as the index language in the field of nuclear science", American Institute of Physics, AIP/UDC7, New York, 1968.

Gains, R. S., and Lee, C. Y., "An improved cell memory", *IEEE Transactions on Computers*, pp. 72-75, February, 1965.

Gallaire, Herve, Minker, Jack, and Nicolas, J. M., "An overview and introduction to logic and data bases", in Gallaire, Herve and Minker, Jack, *Logic and Data Bases*, New York, Plenum Press, 1978.

Healy, L. D., Lipovski, G. J. and Doty, K. L., "The architecture of a context addressed segment-sequential storage" *Proceedings of the AFIPS 1972 Full Joint Computer Conference*, AFIPS Press, Montvale, New Jersey, pp. 691-701, 1972.

Higbie, L. C., "The OMEN computers: Associative array processors", *IEEE COMPCON*, pp. 287-290, 1972.

Higbie, L. C., "Supercomputer architecture", *Computer*, pp. 48-58, December, 1973.

Hillman, D. J., "Negotiation of inquiries in an on-line retrieval system", *Information Storage and Retrieval*, vol. 4, no. 2, 1968.

Hoel, Paul G., Port, Sidney C. and Stone, Charles J., *Introduction to Statistical*

Theory, Boston, Houghton Mifflin Company, p. 224, 1971.

Houston, N. and Wall, E., "The distribution of term usage in manipulative indexes", *American Documentation*, vol. 15, 1964.

Hsiao, David K., "The architecture of a database computer—A summary", *Proceedings of the Third Workshop on Non-Numeric Processing*, May, 1977.

Hsiao, David K., and Kannan, Krishnamurthi, "The architecture of a database computer – part II: The design of the structure memory and its related processors", Technical Report OSU-CISRC-TR-76-2, Ohio State University, October, 1976.

Hsiao, David K., and Kannan, Krishnamurthi, "The architecture of a database computer – part III: The design of the mass memory and its related components", Technical Report OSU-CISRC-TR-76-3, Ohio State University, December, 1976a.

Hsiao, David K., Kannan, Krishnamurthi, and Kerr, D. S., "Structure memory designs for a data base computer", *Proceedings of the ACM Annual Conference*, pp. 343-350, 1977.

Hsiao, David K. and Madnick, Stuart E., "Data Base Machine Architecture in the Context of Information Technology Evolution", *Proceedings of the Third International Conference on Very Large Data Bases*, October, 1977.

Kaplan, A., "A search memory subsystem for a general purpose computer", *Proceedings of the AFIPS 1963 Fall Joint Computer Conference*, vol. 24, Spartan Books, Inc., Baltimore, Maryland, pp. 193-200, 1963.

Keen, E. M., "Evaluation Parameters", in Salton, G. (ed.), *The SMART Retrieval System*, Prentice Hall, Englewood Cliffs, New Jersey, 1971.

Kent, A., *Textbook on Mechanised Information Retrieval*, Wiley and Sons, New York, 1966.

Kerschberg, L., Oskarahan, Esen A., and Pacheco, J. E. S., "A synthetic English query language for a relational associative processor", *Proceedings of the Second International Conference on Software Engineering*, October, 1976.

Kessler, M. M., "An experimental study of bibliographic coupling between technical papers", M.I.T. Report II-1, November, 1961.

Kessler, M. M., "Bibliographic coupling between scientific papers", *American Documentation*, vol. 12, no. 1, January, 1963.

Kessler, M. M., "Comparison of the Results of Bibliographic Coupling and Analytic Subject Indexing", *American Documentation*, vol. 16, no. 3, July, 1965.

King, D. W., et al., "Statistical indicators of scientific and technical communication", 1960-1980, vol. 1, Survey Report to NSF, King Research Inc., DSI, 1976.

Kochen, M., *Principles of Information Retrieval*, Wiley and Sons, Los Angeles, 1974.

Lancaster, F. W., "Evaluation of on-line searching in MEDLARS (AIM-TWX) by biomedical practitioners", University of Illinois, Graduate School of Library Science, Occasional paper No. 101, 1972.

Lancaster, F. W., and Fayen, E. G, *Information Retrieval On Line*, Wiley & Sons, Los Angeles, 1973.

Lang, T., Nahouraii, E., Kasuga, K. and Fernandez, E. B., "An architectural extension for a large database system incorporating a processor for disk

search", *Proceedings of the Third International Conference on Very Large Data Bases*, Tokyo, Japan, October, 1977.

Lee, C. Y., "Intercommunicating cells as a basis for a distributed logic computer", *Proceedings of the AFIPS 1962 Fall Joint Computer Conference*, Spartan Books Inc., Baltimore, Maryland, pp. 130-136, 1962.

Lee, C. Y., and Paull, M. C., "A content adressable distributed logic memory with applications to information retrieval", *Proceedings of the IEEE*, pp. 924-932, June, 1963.

Lin, Chyuan Shiun, "Sorting with associative secondary storage devices", *Proceedings of the National Computer Conference*, pp. 691-695, 1977.

Lin, Chyuan Shiun, and Smith, Diane C. P., "The design of a rotating associative array memory for a relational data base management application", *Proceedings of the International Conference on Very Large Data Bases*, vol. 1, no. 1, September, 1975.

Lin, Chyuan Shiun, Smith, Diane C. P., and Smith, John Miles, "The design of a rotating associative memory for relational database applications", *ACM Transactions on Database Systems* vol. 1, no. 1, pp. 53-65, March 1976. (Revised version of Lin and Smith [1975], cited above).

Linde, Richard R., Gates, Roy and Peng, Te-Fu, "Associative processor applications to real time data management", *Proceedings of the National Computer Conference*, 1973.

Lipovski, G. J., "The architecture of a large distributed logic associative memory", National Technical Information Service, AD 692195, July, 1969.

Lipovski, G. J., "The architecture of a large associative processor", *Proceedings*

of the AFIPS Spring Joint Computer Conference, pp. 385-396, 1970.

Lipovski, G. J., "Architectural features of CASSM: A context addressed segment sequential memory", *Proceedings of the Fifth Annual Symposium on Computer Architecture*, Palo Alto, California, pp. 31-38, April, 1978.

Lockheed Aircraft Corporation, *An evaluation of information retrieval systems*, Burbank, Calif., 1959.

Lowenthal, Eugene I., "A survey - The application of data base management computers in distributed systems", *Proceedings of the Third International Conference on Very Large Data Bases*, Tokyo, Japan, October, 1977.

Luhn, H. P., "Keyword in-context index for technical literature (KWIC Index)", Presented at American Chemical Society, Division of Chemical Research, Atlantic City, N.J., 1959.

Macmillan Information, *Thesaurus of ERIC Descriptors*, seventh edition, New York, Macmillan Publishing Co., Inc., 1977.

Marron, B., "A study of six university-based information systems", Technical Note 781, National Bureau of Standards, Washington, D.C., 1974.

Marron, B., et al., "A mechanized information services catalog", Technical Note 814, National Bureau of Standards, Washington, D.C., 1974.

Martin, T. H., "A feature analysis of interactive retrieval system", Report SU-COMM ICR-74-1, Institute for Communication Research, Stanford University, Stanford, California 94305, 1974.

Martin, T. H., Carlisle, J., and Treu, S., "The user interface for interactive bibliographic searching: an analysis of the attitudes of nineteen information scientists", *Journal of the American Society for Information Science*, vol. 24,

no. 2, March/April, 1973.

McGregor, D., Thompson, R. and Dawson, W., "High performance hardware for database systems", in *Systems for Large Data Bases*, Lockman and Neuhold, eds., Amsterdam, The Netherlands, North Holland, pp. 103-116, 1976.

Melton, J. L., "The semantic code today", *American Documentation*, vol. 13, 1962.

Meister, F. and Sullivan, D. J., "Evaluation of user reactions to a prototype on-line information retrieval system", NASA CR-918, Washington, D.C.

Minker, Jack, "Information storage and retrieval: A survey and functional description", *SIGIR Forum*, a publication of the Special Interest Group on Information Retrieval of the Association for Computing Machinery, vol. XII, No. 2, 1977.

Montgomery, C. and Swanson, D. R., "Machine-like indexing by people", *American Documentation*, vol. 13, 1962.

Mooers, C. N., "Information retrieval on structured content", in *Third London Symposium on Information Theory*, London, 1956.

Minsky, N., "Rotating storage devices as partially associative memories", *Proceedings of the AFIPS 1972 Fall Joint Computer Conference*, AFIPS Press, Montvale, New Jersey, pp. 587-595, 1972.

Moulder, Richard, "An implementation of a data management system on an associative processor", *Proceedings of the National Computer Conference*, 1973.

Murtha, J. C., and Beadles, R. L., "Survey of the highly parallel information processing systems", Office of Naval Research Report No. 4755, November, 1964.

Nakano, R., "A simulator for a RAP virtual memory system", M.S. thesis, University of Toronto, 1976.

National Science Foundation, *Nonconventional technical information systems in current use*, No. 3, Washington, 1962.

Nisenoff, N., Allen, W. and Clayton, A. (Forecasting International Ltd.), "Costs and benefits of some alternative information delivery systems of 1985", final report on NSF grant DSI 76-12161, 1977.

Ozkarahan, Esen A., "An associative processor for relational data bases—RAP", Ph.D. Dissertation, University of Toronto, 1976.

Ozkarahan, Esen A., and Schuster, Stewart A., "A high level machine-oriented assembler language for a data base machine", Technical Report CSRG-74, Computer Systems Research Group, University of Toronto, October, 1976.

Ozkarahan, Esen A., Schuster, Stewart A., and Sevcik, K. C., "Performance evaluation of a relational associative processor", ACM TODS, vol. 2, pp. 175-195, June, 1977.

Ozkarahan, Esen A., Schuster, Stewart A., and Smith, Kenneth C., "A data base processor", Technical Report CSRG-43, Computer Systems Research Group, University of Toronto, Sept. 1974.

Ozkarahan, Esen A., Schuster, Stewart A., and Smith, Kenneth C., "RAP—An associative processor for data base management", *Proceedings of the AFIPS National Computer Conference*, vol. 44, pp. 379-387, 1975.

Ozkarahan, Esen A., and Sevcik, K. C., "Analysis of architectural features for enhancing the performance of a data base machine", ACM TODS, vol. 2, pp.

297-316, December, 1977.

Parhami, B., "A highly parallel computing system for information retrieval", *Proceedings of the AFIPS 1972 Fall Joint Computer Conference*, AFIPS Press, Montvale, New Jersey, pp. 681-690, 1972.

Parker, E., "Behavioral research in the development of a computer-based information system", in Nelson, C. E., and Pollock, D. K. (eds.), *Communication Among Scientists and Engineers*, Heath, Lexington, Mass., 1970.

Parker, J. L., "A logic per track retrieval system", *Proceedings of the IFIP 1971 Congress*, vol. 1, North-Holland Publishing Co., Amsterdam, The Netherlands, pp. 711-716, 1971.

Perry, J. W., Kent, A. and Berry, M. M., *Machine Literature Searching*, New York, 1956.

Reintjes, J. F. and Marcus, R. S., "Interim report on research in the coupling of interactive information systems", Massachussetts Institute of Technology, 1974.

Reiter, R., "An approach to deductive question-answering", BBN Report No. 3649, Bolt, Beranek and Newman, Inc., Cambridge, Mass., Sept., 1977.

Rocchio, J. J., "Evaluation viewpoints in document retrieval", Information Storage and Retrieval, Report ISR 9 to the National Science Foundation, Section XXI, Harvard Computation Laboratory, Cambridge, Mass., 1965.

Rocchio, J. J., "Document Retrieval Systems - Optimisation and Evaluation", Doctoral thesis, Report ISR-10 to the National Science Foundation, Harvard Computation Laboratory, Cambridge, Mass., 1966.

Rosenberg, V., "Technique for monitoring user behavior at the computer terminal interface", *Journal of the American Society for Information Science*, 1973.

Rudolph, J. A., "A production implementation of an associative array processor: STARAN", *Proceedings of the AFIPS 1972 Fall Joint Computer Conference*, vol. 41, pt. 1, AFIPS Press, Montvale, New Jersey, pp. 229-241, 1972.

Rux, P. T., "A glass delay line content-addressable memory", *IEEE Transactions on Computers*, pp. 512-520, 1969.

Salton, G., "The evaluation of automatic retrieval procedures – selected test results using the SMART system", *American Documentation*, vol. 16, no. 3, 1965.

Salton, G., "The evaluation of Computer-Based Information Retrieval Systems", *Proceedings of the 1965 International FID Congress*, Spartan Books, New York, 1966.

Salton, G., *Automatic Information Organisation and Retrieval*, McGraw-Hill, New York, 1968.

Schultz, C. K., "A computer analysis of the Merck Sharp and Dohme Indexing System", Remington Rand Univac, Philadelphia, 1959.

Schuster, Stewart A., Nguyen, H. B., Ozkarahan, Esen A., and Smith, Kenneth C., "RAP.2—An associative architecture for data bases and its applications", *IEEE Transactions on Computers*, vol. c-28, no. 6, June 1979. (Revised version of "RAP.2—an Associative Processor for Data Bases", *Proceedings of the Fifth Computer Architecture Symposium*, May, 1978.)

Schuster, Stewart A., Ozkarahan, Esen A., and Smith, Kenneth C., "A virtual memory system for a relational associative processor", *Proceedings of the*

AFIP'S National Computer Conference, vol. 45, pp. 855-862, 1976.

Seidon, H. R., "A comparative analysis of interactive storage and retrieval systems", Santa Monica, California, System Development Corporation, 1970.

Shaw, David Elliot, "Structure and Abstraction in a System for Conceptual Matching", Stanford Systems Corporation Technical Memo SSC-IR-7702-01, August, 1977.

Shaw, David Elliot, "A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives", Stanford Computer Science Department Report STAN-CS-79-778, October, 1979.

Shaw, David Elliot, "A Relational Database Machine Architecture", *Proceedings of the 1980 Workshop on Computer Architecture for Non-Numeric Processing*, Asilomar, California, March, 1980. (Will also appear in publications of ACM SIGARCH, SIGIR and SIGMOD.)

Shooman, W., "Parallel computing with vertical data", *Proceedings of the 1960 Eastern Joint Computer Conference*, New York, pp. 393-400, 1960.

Slotnick, D. L., "Logic per track devices", *Advances in Computers*, vol. 10, Academic Press, New York, pp 291-296, 1970.

Su, Stanley Y. U., Chen, W. F. and Emam, Ahmed, "CASAL: CASSM's assembly language", Technical Report 7778-7, Computer and Information Sciences Department, University of Florida, March, 1978.

Su, Stanley Y. U., Copeland, George P., and Lipovski, G. J., "Retrieval operations and data representations in a content-addressed disc system", *Proceedings of the International Conference on Very Large Data Bases*, Framingham, Massachusetts, September, 1975.

Su, Stanley Y. U., and Emam, Ahmed, "CASDAL: CASSM's data language", *ACM Transactions on Database Systems*, vol. 3, no. 1., pp. 57-91, March, 1978.

Summit, R. K., "Remote information retrieval facility", NASA CR-1318, Lockheed Missles and Space Co., Palo Alto, 1968.

Swets, J. A., "Information retrieval systems", *Science*, vol. 141, 1963.

Swets, J. A., "Effectiveness of information retrieval methods", *American Documentation*, vol. 20, No. 1, 1969.

Thurber, Kenneth J. and Wahl, Leon D., "Associative and parallel processors", *Computing Surveys*, vol. 7, No. 4, December, 1975.

Trabb-Pardo, Luis, *Set Representation and Set Intersection*, Ph.D. thesis, Report STAN-CS-78-681, Computer Science Department, Stanford University, December, 1978.

Treu, S., "A conceptual framework for the searcher-system interface", in Walker, D. E. (ed.), *Interactive Bibliographic Search: The User/Computer Interface*, Montvale, N.J., AFIP'S Press, 1971.

Verniub, Carlo (Commission of the European Communities), talk delivered at the October, 1978 meeting of EUDISIC, as reported in the News section of Online Review, vol. 2, no. 4, 1978.

Vickery, B. C., *On Retrieval System Theory*, Butterworths, Washington, 1965.

Walker, D. E. (Ed.), *Interactive Bibliographic Search: The User/Computer Interface*, Montvale, N.J., AFIP'S Press, 1971.

Welch, N. O., "A survey of five on-line retrieval systems", Report MTP-322 Mitre Corporation, Washington, D. C., 1968.

Wiederhold, Gio, *Database Design*, McGraw-Hill, pp. 292-294, 1977.

Williams, M. E., "Use of Machine-Readable Data Bases", in Cuadra, C. and Luke, A. W. (Ed.) *Annual Review of Information Science and Technology*, Volume 9, American Society for Information Science, Washington, D. C., 1974.

Wilson, P., "Situational relevance", *Information Storage and Retrieval*, vol. 9, no. 8, 457-471, August, 1973.

Yau, S. S., and Fung, H. S., "Associative processor architecture—a survey", *Computing Surveys*, vol. 9, no. 1, March, 1977.

Young, F. H., "Circulating associative memories", Department of Mathematics Report, Oregon State University, 1962.