American Journal of Computational Linguistics Microfiche 76

NEWSLETTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS

VOLUME 15 - NUMBER 3 JUNE 1978

AMERICAN JOURNAL OF COMPUTATIONAL LINGUISTICS is published by the Association for Computational Linguistics.

SECRETARY-TREASURER: Donald E. Walker, SRI International, Menlo Park, California 94025

EDITOR: David G. Hays, 5048 Lakeshore Road, Hamburg, New York, 14075

ASSOCIATE EDITOR: George E. Heidorn, IBM Research Center, P.O. Box 218, Yorktown Heights, New York 10598

EDITORIAL ASSISTANT: William Benzon

American Journal of Computational Linguistics *Microfiche 76: 2*

CONTENTS

TINLAP-2: PROGRAM AND ABSTRACTS

JULY 25 - 27

UNIVERSITY OF ILLINOIS AT URBANA CHAMPAIGN

TINLAP-2 will consist of six sequential sessions, each of which will address questions of current theoretical interest and questions on long-range research directions. In each session researchers from artificial intelligence, linguistics, psychology, and philosophy will focus their points of view on a particular topic (see schedule below).

Proceedings will be available before the meeting. Each author will give a 10-15 minute presentation (which may include a critique of other papers, an amplification of points in the written paper, etc.) followed by a 90 minute discussion period where questions and comments from the audience will be welcome.

There will be other interesting events during and after the workshop, including the ACL Annual Meeting, a banquet, several opportunities for informal discussions, and events associated with the Linguistic Institute, to be held at the University of Illinois this summer. The LSA (Linguistic Society of

America) meeting will be held at the University of Illinois immediately after TINLAP-2, July 28-30. Information about the LSA meeting can be obtained from Professor Braj Kachru, Department of Linguistics, University of Illinois.

The program for TINLAP-2 is listed immediately below The frame number for the abstract is given in parentheses. Presentations for which no abstract was available are designated with an asterisk.

P R O G R A M

July 24 7:00 pm Reception and Registration at Levis
9:00 pm Faculty Center; Snacks and Cash Bar

July 25 9:00 am LANGUAGE REPRESENTATION AND PSYCHOLOGY
11:45 am

Chair: Dedre Gentner, BBN (7)

Panelists:

David Rumelhart, University of California, San Diego*

Roger Schank, Yale*

Leonard Talmy, Neuropsychiatric Institute of Los Angeles, UCLA

Terry Winograd, Stanford and Xerox PARC

William Woods, BBN*

1:30 pm- LANGUAGE REPRESENTATION AND REFERENCE
4:15 pm Chair: Bonnie Lynn Webber, BBN (10)

Panelists:

hn Anderson, Yale (11)

July 25 (Representation and Reference)

Herbert Clark, Stanford (13)

Andrew Ortony, University of Illinois (14)

Barbara Partee, University of Massachusetts (15)

Candace Sidner MIT (16)

5:00 pm Informal Discussion; Cash Bar and Snacks;
7:00 pm Levis Faculty Center

July 26 9:00 am- DISCOURSE: SPEECH ACTS AND DIALOGUE
11:45 am Chair: Barbara Grosz, SRI International (17)

Panelists:

Joseph Grimes. Cornell (18)

Jerry Morgan, University of Illinois (19)

David Olson, Toronto (20)

Raymond Perrault, Toronto*

Andee Rubin, BBN (21)

1:30 pm - LANGUAGE AND PERCEPTION
4:15 pm Chair: David Waltz, University of Illinois (23)

anelists:

Ruzena Bajcsy, University of Pennsylvania (24)

Ray Jackendoff, Brandeis (26)

Stephen Kosslyn, Harvard*

Zenon Pylyshyn, University of Western Ontario (27)

Yorick Wilks, University of Essex (28)

TINLAP-2

July 26 5:00 pm - ACL ANNUAL MEETING

6:00 pm

6:30 pm Banquet (optional)

Speaker JON ALLEN, M.I T

July 27 9:00 am - INFERENCE MECHANISMS IN NATURAL LANGUAGE

11.45 am Chair. Aravind Joshi, University of Pennsylvania**

Panelists:

Eugene Charniak, Yale (29)

Allan Collins, Yale (30)

Jerrold Kaplan, University of Pennsylvania (31)

Raymond Reiter, University of British Columbia (32)

Charles Rieger, University of Maryland (34)

Stuart Shapiro, SUNY Buffalo (35)

Rand Spiro, University of Illinois (36)

1:30 pm - COMPUTATIONAL MODELS AS A VEHICLE FOR

4:15 pm THEORETICAL LINGUISTICS

Chair· Ronald Kaplan, Xerox PARC*

Panelists.

Joseph Grimes, Cornell*

Mark Liberman, Bell Laboratories*

Mitch Marcus, MIT*

Tom Wasow, Stanford*

**No paper to be presented.

TESTING THE PSYCHOLOGICAL REALITY OF A REPRESENTATIONAL MODEL

Dedre Gentner

Bolt Beranek and Newman, Inc.

A research program is described in which a particular representational format for meaning is tested as broadly as possible. In this format, developed by the LNR research group at The University of California at San Diego, verbs are represented as interconnected sets of subpredicates. These subpredicates may be thought of as the almost inevitable inferences that a listener makes when a verb is used in a sentence. They confer a meaning structure on the sentence in which the verb is used. To be psychologically valid, these representations should capture (at least):

1. Similarity of meaning:
   The more similar two verbs seem in meaning to people, the more their representations should overlap.

2. Confusability:
   The more confusable two verb meanings are for people, the more their representations overlap.

3. Memory for sentences containing the verb:
   The sentence structures set up by the verb's meaning should in part determine the way in which sentences are remembered.

4. Semantic integration:
   The representations should allow for the integration of information from different sentences into discourse structure.

5. Acquisition patterns:
   The structural partitions in the representations should correspond to the structures children acquire when they are learning the meanings of the verbs

6. Patterns of extension:
   The representations should be extendable so as to reflect the ways in which people interpret verb meanings when the verbs are used outside their normal context.

7. Reaction times:
   The time taken to comprehend a sentence using a given verb should reflect the structural complexity of the verb meaning.

Experiments concerned with predictions 1 - 5 are described here. The results are promising for a general approach of representation of meaning in terms of interrelated subpredicates, but do not clearly distinguish between several similar representations. For example, to test prediction (2), I read people sentences containing verbs with similar meanings, and asked them to recall the sentences. The degree of overlap in the semantic structures was a good predictor of the number of confusions between sentences. In another sentence-memory experiment (prediction (3)) semantically complex verbs that provided more underlying interconnections between the nouns in a sentence led to better memory for the nouns in the sentence than simple general verbs, or than other complex verbs that did not provide such extra interconnections. To test prediction (5), I tested children's comprehension of a set of possession verbs. Both the order of acquisition among the verbs and the kinds of errors fitted well with an account of the acquisition of verb meaning in terms of interconnected subpredicates

This research illustrates a breadth-first approach to testing a representation. In the breadth-first approach, many different psychological predictions are made. Each different area of prediction requires a set of process assumptions, and in each case the process assumptions used are those that seem most plausible given previous research in the field. If one representational format can make correct predictions about a number of different kinds of psychological phenomena, then that representation stands a greater chance of being generally useful than one which was tested in only one depth-first way.

Leonard Talmy
Neuropsychiatric Institute, UCLA

A sentence (or other portion of discourse) is taken to evoke in the listener a meaning complex, here called a "cognitive representation" The lexical elements of the sentence seem, by and large, to specify the content, or substance, of the cognitive representation, while the grammatical elements specify its structure. Thus, looking systematically at the actual notions specified by grammatical elements can give us a handle for ascertaining the very makeup of (linguistic-) cognitive structuring. We accordingly examine a number of grammatically specified notions, observe the systems or categories in which they pattern, and speculate on broader cognitive connections.

Some provisional findings have already emerged: Grammatical specifications for structure are preponderantly relativistic or topological, and exclude the fixed or metrically Euclidean. The systems in which grammatical notions pattern include:

plexity (uniplex/multiplex)	degree of extensionality
state of boundedness	pattern of distribution
state of dividedness	axial characteristics
level of synthesis	perspectival characteristics
level of exemplarity	scene-breakup characteristics

Grammatical specification of structuring appears, in certain abstract characteristics, to be isomorphic with the structuring of visual perception.

Reference:

Talmy, L. Rubber-Sheet Cognition in Language. In: Papers from the 13th Regional Meeting, Chicago Linguistic Society. W. Beach, et. al., eds. University of Chicago. 1977.

Description Formation and Discourse Model Synthesis

Bonnie Lynn Webber
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, MA 02138

Researchers in linguistics, psychology and artificial intelligence have recently begun to abandon a purely linguistic approach to definite anaphora (definite pronouns and noun phrases. Instead they posit the notion of reference into a model that a listener/reader is synthesizing from the discourse: the referent of a definite anaphor is then not a linguistic object, but rather an entity in a model. Such a model has been called a "world of discourse" [levin & Goldman, 1978]; a "universe of discourse" [lyons 1978], a"discourse model" [Nash-Webber 1977; Webber 1978] and a "domain of interpretation" [Stenning 1975], inter alia. Its synthesis is what interests me.

Discourse model synthesis intuitively seems to result from interactions between the listener/reader's expectations and various features of the text. What these interactions are is not clear. A discussion of how the listener reader's changing expectations can affect discourse model synthesis can be found in [Collins, Brown & Larkin, 1977]. What I shall discuss here are some features of the text that affect what entities appear in a discourse model and how such entities are described. In the course of presenting these features, I will argue that having an appropriate description for a discourse entity is critical to its successful reference later on. I will then argue that recognizing formal aspects of the text is critical to the formulation of appropriate descriptions. While this is not a sufficient condition for successful reference, it is certainly a necessary one.

References

Collins, A., Brown, J.S. & Larkin, K. (1977)
Inference in Text Understanding. Technical Report No. 40, Center for the Study of Reading, University of Illinois and Bolt Beranek & Newman Inc. December 1977.

Levin J. & Goodman, N. (1978)
Process Models of Reference. Unpublished MS., Information Sciences Institute, Marina Del Rey CA.

Lyons, J. (1977)
Semantics. England: Cambridge University Press, 1977.

Nash-Webber, F.L. (1977)
Inference in an Approach to Discourse Anaphora. Technical Report No. 77, Center for the Study of Reading, University of Illinois and Bolt Beranek & Newman Inc. December 1977.

Stenning, K. (1975)
Understanding English Articles and Quantifiers. Unpublishted Doctoral Dissertation, the Rockefeller University, 1975.

Webber, B.L. (1978)
A Formal Approach to Discourse Anaphora. Technical Report No. 2761, Bolt Beranek & Newman Inc., Cambridge MA April 1978.

Representation of Individuals in Semantic Nets

John Anderson
Yale University

Abstract

Research is reported concerned with how subjects process multiple referring expressions. In one experiment, subjects learn sentences such as:

The smart Russian cursed the salesgirl
The smart Russian rescued the kitten
The tall lawyer adopted the child
The tall lawyer caused the accident

and only later learn that the smart Russian is the same person as the tall lawyer. How do subjects integrate the information about the smart Russian with information about the tall lawyer? It is information subjects have set up two nodes in memory, one for each definite description. Upon learning of the identity of the two descriptions, they introduce into memory a proposition indicating the identity of the two individual nodes. They also start a process of copying information from one node to the other node. In effect, they choose to abandon one of the nodes.

It is argued that a similar process occurs when subjects recognize the referent of a definite description--but on a much shorter time scale. So, suppose a subject hears:

The first president of the United States was a bad husband.

The proposal is that the subject creates a new node to represent the subject of that sentence, attaches to this node network structure to encode it is the first president of the United States, uses this network structure to guide a search of memory for the referent, finds a node corresponding to George Washington (GW), indicates that the new node and the GW node are the same, copies fromthe new node to

the GW node the bad husband predicate, and abandons the new node. Data is presented consistent with this process model for dealing with the referents of definite descriptions.

Reference Diaries

Herbert H. Clark and Catherine Marshall
Stanford University
Standford CA

Speakers and listeners are forced to keep diaries about what they know about each other because, to use or interpret a definite reference, they have to assess the knowledge they "share" with each other about the thing being referred to. More precisely, it can be shown that the speaker and listener have to assess what is technically called their mutual knowledge about the referent. This, however, raises a striking paradox. The assessment of mutual knowledge logically requires an infinity of separate tests, and if each test takes a finite amount of time, then people would take an infinite length of time to make or interpret any definite reference. As a solution to this problem, we argue, people use the heuristic of searching their diaries for an event that satisfies a condition we call triple co-presence. With such an event they can satisfy the infinity of tests required by mutual knowledge in a single step. We discuss the kinds of events that satisfy tripble co-presence, and we provide experimental evidence that when people cannot find such an event they are open to error in their interpretation of definite reference.

Some pragmatic constraints on the construction and interpretation of definite descriptions.

Andrew Ortony
University of Illinois at Urbana-Champaign

Abstract

Both the production and the comprehension of definite descriptions requires that inferences be made. In many cases the inferences are trivial and of little theoretical importance or interest. However, there is a class of definite descriptions that have the characteristic that their relation to their antecedents depends on pragmatic inferences (contrasted with deductively logical inferences). In such cases, the predicate underlying the definite description cannot be taken to be true of the antecedent as a result of any entailment relations. Rather, the predicate is taken as being probabilistically related. This paper examines this class of "pragmatic definite descriptions" more closely, paying particular attention to what constrains the set of candidate descriptions that can be used to refer to the antecedent. One of the results of this examination is the postulation of a theory about the extent to which an indirect speech act can be indirect.

Bound Variables and Other Anaphors

Barbara H. Partee

ABSTRACT

The aim of the paper is to delimit a subset of pronominal anaphora for which the logicians notion of bound variables gives the best account. It can be argued that some cases of anaphora must be viewed this way and some cases cannot be. The clearest cases of bound variable anaphora involve antecedents like every man and no man which are singular in form but do not refer to individuals. But even with an antecedent like John, an anaphoric pronoun must sometimes be viewed as a bound variable to account for one of the readings (the so-called "sloppy identity" reading) of (1):

(1) John was sure he would win, and so was Bill.

Bound variable anaphora will be contrasted with free "discourse" anaphora; the differences between them suggest that the former is essentially a semantic phenomenon, the latter largely pragmatic.

The Use of Focus as a Tool for Disambiguation of Definite Noun Phrases

Candy Sidner
MIT AI Lab

This paper will center on a discussion of the use of focus in the interpretation of anaphoric noun phrases in discourse. The need for focus will be discussed, and a description of focus shifting will be given. Focus provides a means of representing the central concept of a discourse. The ways in which a definite noun phrase, specific or generic, can be used are constrained by its relation to the focus and by the ways in which the focus can be shifted. The discussion of anaphoric defnps will present a taxonomy of cases, distinguished by the relation of the defnp to the focus. This taxonomy inclues several kinds of inference dependent cases. The paper will concentrate discussing on the process of understanding defnps, and will present rules governing the ways a defnp can be used so that the hearer/reader can understand its co-referent. This paper will also distinguish reference, co-reference and internal reference, and point out the need for these distinctions in natural language research.

FOCUSING IN DIALOG

Barbara J. Grosz
SRI International
Menlo Park, California

When two people talk they focus on only a small portion of what each of them knows or believes. Both what gets said and how it gets interpreted depend on this narrowing of attention to a common highlighted portion of knowledge. One of the effects of understanding an utterance is to become focused on certain entities (relationships and objects) and on particular views of those entities. A speaker provides a hearer with clues to what to look at and how to look at it -- what to focus on, how to focus on it, and how wide or narrow that focus should be. These clues may be linguistic or they may come from knowledge about the relationships among entities in the domain (the structure of the things being talked about) or from the environment in which the dialog occurs. Linguistic cues may be either explicit, given directly by certain words, or implicit, deriving from sentential structure or from rhetorical relationships between sentences.

This paper examines focusing in dialog, discusses an initial representation in which focusing is based on domain structure cues, and examines from this perspective what other information and models are needed to extend the formalization of focusing to more general dialogs. The importance of focusing is illustrated by considering its role in the processes of understanding and generating definite descriptions.

TOPIC LEVELS

Joseph E. Grimes
Cornell University
Ithaca, N.Y.

In order to interpret either a dialogue or a monologue, some referential elements must be agreed on by the speaker and the hearer as a starting point. This is the topic in the sense proposed by Searle and Gundel.[1] Even though the topic normally shifts away from its starting point in the course of a text, whatever is being treated as topic in a particular part of the text receives special treatment in determining the expression to be used.

Two levels of topic, global and local, in English conversation have been noted by Grosz. They imply different strategies for establishing the reference of pronouns. It is useful to consider them in the light of two other languages, Longuda of Nigeria and Bacairi of Brazil, that distinguish topic from nontopic by their pronoun systems.

Finally, there is some evidence from both Greek and English that there may be more than two topic levels operating simultaneously in nonconversational texts.

ABSTRACT

Toward a rational model of discourse comprehension

J. L. Morgan
Center for the Study of Reading
and
Department of Linguistics
University of Illinois

Models of discourse or text often treat connected discourse in a manner analogous to the treatment of sentences in traditional and generative grammar; i.e. as a formal object to be decoded by means of certain formal operations. I point out in this paper that even where this view is not explicitly proposed, it is often implicit. Against this common view I argue that the only kind of discourse model that is likely to succeed is one that is built around two important hypotheses: first, that the key to discourse comprehension is the attempt to infer the details of the plan that the speaker/writer follows in constructing the text; second, that a large portion of the work of a discourse comprehension model should be derived from a theory of practical reasoning. I will sketch the outline of a model (or more accurately, a schema for a large class of possible models) that incorporates these suggestions, pointing out the role of practical reasoning processes, and arguing that notoriously confused notions like "given/new" and "expected information" can only be made sense of in such a model.

SOME SOCIAL AND LOGICAL ASPECTS OF MEANING IN THE LANGUAGE OF SCHOOL-AGED CHILDREN

David R. Olson
Ontario Institute for Studies in Education
Toronto, Canada

It is conventional to treat the meaning of an utterance in a discourse in terms of two components: the propositional component and the pragmatic or speech act component, the first indicating the meaning of the sentence, the second indicating its intended use by the speaker. I shall present some arguments and evidence that these two systems are interdependent. Roughly, it appears that social considerations, primarily status, determine which aspects of a proposition are lexicalized in the utterance. Thus, a child with high status relative to his interlocutor may use a command, "Give me a block", while if he has low status relative to his interlocutor he may use a request, "May I have a block?" If he is an equal, a peer, (and perhaps only then) he will use an explicit true proposition such as, "You have two more than me." Only in this third case is the propositional meaning explicit in the sentence per se, and only in this case is an affirmative or negative response dependent strictly upon truth conditions (rather than compliance, for example).

This conception of the social aspects of meaning will be examined through an analysis of what is said vs. what is meant in some child-child and teacher-child conversations.

Paper prepared for Theoretical Issues in Natural Language Processing (TINLAP). Urbana, University of Illinois, July 25-27, 1978.

WHO AM I TALKING TO AND CAN THEY TALK BACK: THE EFFECT OF AUDIENCE AND INTERACTION ON DISCOURSE MODELS

Andee Rubin
Bolt Beranek and Newman Inc.
Cambridge, Mass.

Communication among people occurs in a vast variety of settings from reading a book to participating in a conversation, from listening to a tape to reading a transcript of a lecture. Most discussions of discourse, speech acts and dialogue, however, consider a very particular kind of communicative situation: face-to-face oral conversations between two participants in which there is a common spatial and temporal context. Dialogues between a computer system and a person differ from this model along at least two dimensions: the modality of the interaction (current computer-person dialogues are written) and the lack of spatial commonality, indicated by the impossibility of communicating with gestures and facial expressions. The implications of these differences for theories of discourse are poorly understood. Worse yet, they illustrate only a small subset of the dimensions along which language experiences may vary. What relevance do the theories we advance to account for these interchanges have for other communicative experiences such as listening to a lecture or reading a play?

This paper will focus on two other aspects of language experience which have consequences for the dialogue models we build: audience and the degree of interaction. In both situations described above, the audience is a single other person (or system) and interaction between the participants - or even interruption - is immediate. But in a book, for example, the audience is large and not well defined and the book's reader must adopt new strategies to compensate for the fact that interaction is impossible. In a personal letter, on the other hand, the audience is a single other person, similar to the conversational situation. Interaction, however, is impossible or at least attenuated; the reader can obtain clarifying information, but the time lapse will be significant.

I will consider in this paper where various language experiences lie along these two dimensions and what the implications of these differences are for models of discourse and dialogue.

On the Interdependence of Language and Perception

David L. Waltz
Coordinated Science Laboratory
University of Illinois at Urbana/Champaign

Without a connection to the real world via perception, a language system cannot know what it is talking about. Similarly, a perceptual system must have ways of expressing its outputs via a language (spoken, written, gestural or other). The relationship between perception and language is explored, with special attention to what implications results in language research have for our models of vision systems, and vice-versa. It is suggested that early language learning is an especially fertile area for this exploration. Within this area, we argue that perceptual data is conceptualized prior to language acquisition according to largely innate strategies, that this conceptualization is in terms of an internal, non-ambiguous "language," that language production from its beginnings to adulthood is a projection of the internal language which selects and highlights the most important portions of internal concepts, and that schemata produced in the sensory/motor world are evolved into schemata to describe abstract worlds. Examples are provided which stress the importance of "gestalt" (figure-ground) relationships and projection (3-D to 2-1/2 or 2-D , conceptual to linguistic, and linguistic to conceptual); finally mechanisms for an integrated vision-language system are proposed, and some preliminary results are described

The Problem of Naming Shapes: Vision-Language Interface

by

R. Bajcsy*
and
A.K. Joshi*

Computer and Information Science Department
University of Pennsylvania
Philadelphia, PA 1904

1. Introduction

In this paper, we will pose more questions than present solutions. We want to raise some questions in the context of the representation of shapes of 3-D objects. One way to get a handle on this problem is to investigate whether labels of shapes and their acquisition reveals any structure of attributes or components of shapes that might be used for representation purposes. Another aspect of the puzzle of representation is the question whether the information is to be stored in analog or propositional form, and at what level this transformation from analog to propositional form takes place.

In general, shape of a 3-D compact object has two aspects: the surface aspect, and the volume aspect. The surface aspect includes properties like concavity, convexity, planarity of surfaces, edges, and corners. The volume aspect distinguishes objects with holes from those without (topological properties), and describes objects with respect to their symmetry planes and axes, relative proportions, etc.

---

* This work has been supported under NSF Grant #MCS76-19465 and NSF Grant #MCS76 19466.

We will discuss some questions pertinent to representation of a shape of a 3-D compact object, without holes, for example: Is the surface aspect more important than the volume aspect?, Are there any shape primitives? In what form are shape attributes stored?, etc. We shall extensively draw from psychological and psycho-linguistic literature, as well as from the recent AI activities in this area.

An Argument Combining Linguistic and Visual Evidence

Ray Jackendoff

Brandeis University

ABSTRACT

The notion from gestalt psychology of a "figure" emerging from a "background" will be shown to be crucially involved in a complete description of the successful communication of so-called "pragmatic anaphora" - uses of pronouns without linguistic antecedents such as that in (1).

(1) I bought that pointing last Saturday.

A survey of types of pragmatic anaphora in English will then be used to show that the notion of "figure" must encompass a much wider range of perceptual entities than commonly assumed. Finally the implications for linguistic semantics, philosophy, perceptual theory, and cognitive theory will be discussed

Language and Perception

Zenon Pylyshyn
University of Western Ontario

A language comprehension system without a perceptual component would, in an important sense, not know what it was talking about even if it could carry on a sensible dialogue. More significantly, a theory of comprehension would be seriously deficient if it did not relate linguistic representations to ones which derive from non-linguistic sources. This bridge is necessary in order to explain how terms refer as well as to explain how language is acquired. This paper will discuss and support the position that natural language learning is only possible because of the prior existence of mentalese --a language-like system of representation for perceptual as well as more abstract conceptual contents. How this comes into being cannot be given as an information processing explanation since it requires an account of the development of the underlying machine architecture--not of its language processing software (i.e., interpreters).

SEMANTIC PRIMITIVES IN LANGUAGE AND VISION

Yorick Wilks

Department of Language and Linguistics
University of Essex
England

ABSTRACT

An argument is presented that, on the basis of the evidence at present available, there is no reason to believe that the semantic primitives required by natural language understanding have any basis or grounding in vision. And, moreover, whatever may ultimately turn out to be the way we work, there is no reason to believe that trying to ground one sphere of AI on the other language primitives on visual ones, would assist research in either area. A number of systems of primitives are examined briefly in order to strengthen the above argument.

With a Spoon in my Hand this must be the Eating Frame

Eugene Charniak
Department of Computer Science
Yale University

ABSTRACT

A language comprehension program using "frames," "scripts," etc. must be able to decide which frames are appropriate to the text. Often there will be explicit indication ("Fred was playing tennis" suggests the TENNIS frame) but it is not always so easy. ("The steering wheel was hot, but Jack had to be home by 3" suggests DRIVING, but how?) This paper will examine how a program might go about determining the appropriate frame in such cases. The basic idea will be taken over from Minsky (1975) in that it will be assumed that one usually has one or more context frames, so that one only needs worry if information comes in which does not fit them. As opposed to Minsky however the suggestions for new context frames will not come from the old ones, but rather directly from the conflicting information. A major portion of the paper then will be concerned with how we will index context frames (e.g., DRIVING) under the clues which suggest them (e.g., STEERING-WHEEL).

Abstract

Human Plausible Reasoning

Allan Collins
Bolt Beranek and Newman Inc.

The paper outlines a computational theory of human plausible reasoning constructed from analysis of people's answers to everyday questions. Like logic, the theory is expressed in a content-independent formalism. Unlike logic, the theory specifies how different information in memory affects the certainty of the conclusions drawn. The theory consists of a dimensionalized space of different inference types and their certainty conditions, including a variety of meta-inference types where the inference depends on the person's knowledge about his own knowledge. The protocols from people's answers to questions are analyzed in terms of the different inference types. The paper also discusses how memory is structured in multiple ways to support the different inference types, and how the information found in memory determines which inference types are triggered.

Indirect Responses to Loaded Questions

S. Jerrold Kaplan
University of Pennsylvania

ABSTRACT:

Casual users of natural language (NL) systems are typically inexpert not only with regard to the technical details of the underlying programs, but often with regard to the structure and/or content of the domain of discourse. Consequently, NL systems must be designed to respond appropriately when they can detect a misconception on the part of the user. Several conventions exist in cooperative conversation that allow a speaker to indirectly encode their intentions and beliefs about the domain into their utterances, ("loading" the utterances) and allow (in fact, often require) a cooperative respondent to address those intentions and beliefs beyond a literal response. To be effective, NL computer systems must do the same.

This paper will explore several types of indirect responses to NL questions, showing that in the Data Base query domain general computational models exist that can determine both when an indirect response is required and what that response should be. An implementation of these ideas will be presented that demonstrates their immediate practical value in NL systems. This paper will take the position that language related inferences (i.e., inferences driven directly from the phrasing of the question) are to a great extent separable from deeper reasoning and deduction processes, and are sufficient to produce a wide variety of useful and cooperative behavior.

Ray Reiter
University of British Columbia

A B S T R A C T

I propose to discuss a number of principles for structuring knowledge, principles which are motivated by the need for efficient deductive inference in question-answering systems. The notion of structure that I will define is, in some sense, orthogonal to but not antithetical to a number of current ideas in AI regarding the organization of knowledge.

Intensional vs. Extensional Representations of Knowledge

Given a predicate P, we can represent what we know about P extensionally, or intensionally, e.g. as a procedure or a general axiom) or by some combination of both. How should this decision be made? It turns out that if we represent appropriate predicates extensionally then

(i) No infinite deductive searches can arise.

(ii) Certain intensional knowledge becomes irrelevant for deduction and may be discarded.

The Closed World Assumption (CWA)

In domains for which we have perfect knowledge (e.g. blocks worlds) it is appropriate to make the CWA. This means, roughly speaking, that to establish a negative fact, it is sufficient to fail to prove its positive counterpart. The CWA yields a significant decrease in the complexity of deductive reasoning. In addition, it induces a decomposition of the available knowledge into two components, one of which is used only for integrity, and the other only for deductive inference.

Horn Data Bases

It is well known that whenever the knowledge about a domain is representable by Horn formulae (i.e. formulae of the form $P_1 \wedge P_2 \wedge \ldots \wedge P_n \supset P_{n+1}$ where $P_1, \ldots, P_{n+1}$ are positive) then consequent and/or antecedent reasoning is complete for that domain. This result is not true for non Horn domains - more sophisticated

reasoning, such as case analysis, may be required. Another nice feature of Horn domains is that the CWA does not lead to any inconsistencies. Not all domains can be represented by Horn formulae. For some such domains it is possible to render them "essentially" Horn by extensionally representing certain appropriately chosen predicates, in which case all of the virtues of Horn domains may be salvaged.

Summary

I am proposing the following structuring principles:

1. If possible, make the CWA.
2. If the knowledge base is non Horn, make it "essentially" Horn by extensionally representing appropriate predicates.
3. Eliminate infinite deduction paths by extensionally representing certain suitably chosen predicates.
4. Under 1, 2 and 3, certain intensions will no longer be relevant for deduction. Remove these.

Inference and Parsing Architecture in GRIND-1, a Full-Scale Story Comprehender

Chuck Rieger
Computer Science Department
University of Maryland
College Park, Maryland 20742

ABSTRACT: The paper describes the inference and parsing components of GRIND-1, a full-scale story comprehension project based on a Walt Disney Book of the Month Club book, "The Magic Grinder". Topics include: (1) the sense network parser and its interaction with inference, (2) character personality trait modeling via behavioral tags, (3) two-character relationship modeling, and (4) plot representation and plot level prediction. The main areas of emphasis will be on the representation of inference, and on the various types of inference conditioning that stem from the character models and plot.

Path-Based and Node-Based Inference in Semantic Networks*

Stuart C. Shapiro
Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

ABSTRACT

Two styles of performing inference in semantic networks are presented and compared. Path-based inference allows an arc or a path or arcs between two given nodes to be inferred from the existence of another specified path between the same two nodes. Path-based inference rules may be written using a binary relational calculus notation. Node-based inference allows a structure of nodes to be inferred from the existence of an instance of a pattern of node structures. Node-based inference rules can be constructed in a semantic network using a variant of a predicate calculus notation. Path-based inference is more efficient, while node-based inference is more general. A method is described of combining the two styles in a single system in order to take advantage of the strengths of each. Applications of path-based inference rules to the representation of the extensional equivalence of intensional concepts, and to the explication of inheritance in hierarchies are sketched.

---

*Preliminary version of a paper to be presented at "Theoretical Issues in Natural Language Processing," the 1978 annual meeting of the Association for Computational Linguistics, Urbana/Champaign, Illinois, July 25-27, 1978.

Processing of Inferences

Rand Spiro and Joseph Esposito
University of Illinois

Abstract

The hypothesis that pragmatic inferences presented in text are taken for granted, superficially processed, and not stably or enduringly represented in memory was investigated. Stories were read which in some conditions contained information vitiating the implicational force of explicit inferences. The vitiating information was presented either before or after the inferences. In Experiment I, errors in memory for the inferences were prevalent in the "after" but not the "before" condition. Two kinds of errors were made: saying the inference had not been presented in the story; or, if it was remembered as having been presented, altering the specific content of the inference to produce the opposite of what was actually presented. The latter errors produced coherence with the vitiating information, and subjects were not able to differentiate these errors from correct responses. In Experiment II, the results of Experiment I were replicated, and a "spontaneous correction" interpretation was rejected. The results of both experiments combine to support the hypothesis of superficial processing and unstable representation of explicit inferences. The results provide a link between processes occurring at comprehension and recall in the State of Schema model of accommodative reconstruction.

DICTIONARY SOCIETY OF NORTH AMERICA
SPECIAL MEETING

JULY 27, 1978

UNIVERSITY OF ILLINOIS, URBANA

ROOM 407, LEVIS FACULTY CENTER

CONTACT: Dr. Ladislav Zgusta Phone: 217 - 333-3563
Department of Linguistics
4088 Foreign Languages Building
University of Illinois, Urbana
61801

KEYNOTE LECTURE: YAKOV MALKIEL, DEPARTMENT OF LINGUISTICS, UNIVERSITY OF CALIFORNIA, BERKELEY

The Lexicographer as a Mediator Between Linguistics and Society

Other lectures include:

FREDERIC G. CASSIDY, Computer Mapping of Lexical Variants for DARE

JOHN J. NITTI, Computers and the Old Spanish Dictionary

NEIL H. OLSEN, Computational Lexicography at the University of Hawaii - Methods and Applications (participation tentative)

GEORGE FARR (NEH), Funding Possibilities for Lexicographic Work (participation tentative)

HOUSING: Illini Union
University of Illinois
Green Street
Urbana 61801

$15.00 single
$21.00 double

Make reservations directly with the Union.

NCC '79 PERSONAL COMPUTING FESTIVAL

JUNE 5 - 7, 1979, NEW YORK CITY

PRESENT A PAPER
GIVE A TALK
ORGANIZE A PANEL
DELIVER A TUTORIAL

THEME: IS WORTH IT?

IS WHAT WORTH IT? Any and every aspect of personal computing is being questioned

WHAT IS IT WORTH? How is personal computing enriching our individual lives, the lives of our families, and improving the quality of life in general?

IS IT WORTH WHAT? he money, the time, the effort, the acquiring of technical expertise, even the criticism.

Potential participants should send a "letter of intent" as soon as possible, but no later than February 1, 1979 to Jay P. Lucas The letter should include an abstract and a brief biography

PAPERS presented during the program will be published. Potential authors will be mailed a Festival Author's Kit with instructions and materials. Papers must be received by March 15, 1979 in the specified camera-ready format. Authors will be notified by May 1, 1979.

PANELS, TUTORIALS AND TALKS: Session leaders should submit a brief abstract describing either the scope of the proposed session or the tentative title of the presentation by February 1, 1979. The prospective organizer should submit a list of proposed participants, their affiliations, and a brief biography of each.

FESTIVAL CHAIRMAN

Richard Kuzmack
1435 Layman Street
McLean, VA 22101

703 821-2873(home)

JOINT PROGRAM CHAIRMEN

Russell Adams
3008 Mosby Street
Alexandria, VA
22305

701 548-8261(home)

Jay P. Lucas
3409 Saylor Place
Alexandria, VA
22304

703 751-3332 (home)

1979 NATIONAL COMPUTER CONFERENCE

JUNE 4 - 7, NEW YORK CITY

HOW TO PARTICIPATE: Write a paper.
Propose a technical or panel session
Volunteer to be a panelist.
Send ideas for topics.
Suggest special activities.

SUGGESTED AREAS FOR PARTICIPATION:

MANAGEMENT — SCIENCE AND TECHNOLOGY

APPLICATIONS — SOCIAL IMPLICATIONS

GUIDELINES:

PAPERS: Should be previously unpublished. Must be in final form with quality figures and tables. All papers will be refereed. 2500 words to 5000 words. Six copies of the paper should be submitted along with six copies of a title page containing a title, 150 word abstract, 4 to 6 keywords, author's affiliation, telephone number and mailing address.

TECHNICAL OR PANEL SESSIONS: Proposals should include a topic description, suggested session chairpersons and presenters, panelists, and indication or importance of session and anticipated audience.

SEND SUBMISSIONS BY NOVEMBER 1, 1978 TO THE PROGRAM CHAIRMAN.

CONFERENCE CHAIRMAN: Merlin G. Smith
T.J. Watson Research
P.O. Box 218
Yorktown Heights,
New York 10598

PROGRAM CHAIRMAN
Richard E. Merwin
Box 32222
Washington, DC 20007

SHORT NOTICE OF UPCOMING C O N F E R E N C E S

FOURTH JOINT INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION

November 7 - 10, 1978

Kyoto, Japan

Sponsor: IEEE

Contact: Professor Makato Nagao
Department of Electrical Engineering
Kyoto University
Sakyo, Kyoto 606 JAPAN

ACM '78

December 4 - 6, 1978

Washington, D.C

Contact: Richard Austing
Department of Computer Science
University of Maryland
College Park, MD

COMPUTER ELEMENTS WORKSHOP ON PUTTTING A MATURING TECHNOLOGY TO WORK

December 11 - 14, 1978

Mesa, Arizona

Sponsor: IEEE - CS

Contact: S.M. Neville
Bell Labs, Room 2B438
Naperville, IL 60540

CONFERENCES

145th ANNUAL MEETING OF AAAS

January 3 - 8, 1979

Chicago, Illinois

Contact: Dr. Arthur Herschman
1776 Massachusetts Avenue, NW
Washington, DC 20005

Exhibits Only: Dr. Edward Ruffing
Scherago Associates
1515 Broadway
New York, NY 10036

ANNUAL MEETING OF THE COMPUTER LAW ASSOCIATION

March 4, 1979

Washington, D.C.

Contact: Michael Foursnaw
Suite 1100
1776 K St., N.W.
Washington, DC 20006

NFAIS 21st ANNUAL CONFERENCE

March 6 - 7, 1979

Arlington, VA

Contact: Toni Carbo Bearman
NFAIS
3401 Market St.
Philadelphia, PA 19104

ANNUAL CONFERENCE OF THE CANADIAN LIBRARY ASSOCIATION

June 14 - 20

Ottawa, Ontario, Canada

Contact: Business Manager
Canadian Library Association
151 Sparks Street, 9th Floor
Ottawa, Ontario K1P 5E3
Canada

AMERICAN LIBRARY ASSOCIATION ANNUAL CONFERENCE

June 24 - 30, 1979

Dallas, Texas

Contact: American Library Association
50 East Huron Street
Chicago, IL 60611

SIXTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI-79)

August 20 - 24

Tokyo, Japan

Contact: Prof. Bruce Buchanan, Program Chairman
Computer Science Department
Stanford University
Stanford, CA 94305

American Journal of Computational Linguistics *Microfiche 76: 43*

RECOGNITION MEMORY
SEMIONICS ASSOCIATES: TECHNICAL NOTE

Recognition Memory (REM)

Associative Memory

Computer memories may be divided into two basic types, one of which has been almost unknown up to now, except among specialists. In the well-known type stored items are accessed by means of their addresses, the address being a number that identifies the location in which the item is stored. This is the type of memory used in all the well-known computers, throughout the history of electronic data processing. The relatively little-known type has been discussed and dreamed about by various computer scientists for some years now, under the designations Associative Memory and Content-Addressable Memory (CAM). With this type of memory, an item may be accessed simply by being named: if it is present, those locations which have it recognize it and respond. In his recent book on the subject[1] Professor Caxton Foster describes the difference between these two types of memories by comparison with a situation in which a teacher wants to know which students in the class, if any, have a particular book. If he operates like a computer with a conventional memory (and assuming no prior clues that would eliminate certain portions of the class from consideration) he would start with the first seat in the first row and ask the student in that seat, "Do you have this book?" He would then repeat this step for the second seat, and so forth, through every seat in the class. If he operated like an associative memory, on the other hand, he could simply say to the whole class, "Will all the students who have this book please raise their hands." As this illustration suggests, the advantages of associative memory for any kind of recognition or information retrieval or language processing application are rather striking. Why then have computers (with a few rare exceptions) always been built with location accessed memories? There are apparently two major reasons: (1) inertia or the heavy influence of tradition, and doubtless more important, (2) the greater costs involved in building associative memories.

Recognition Memory

Over a period of several years now, various computer scientists have been writing articles in technical journals proposing designs and applications for associative memories[2] and a few experimental models have been constructed, but at such great cost that they have remained experimental. There is now, since 1972, a commercially available system with an associative memory, but it is very expensive and therefore not widely used, nor even widely known. Against this background Semionics Associates, a one-man company, has developed a type of system called Recognition Memory (REM) which embodies some inventions (patents pending) that make associative memories economical to construct. While lacking some of the advantages of more elaborate associative memories, REM has their essential properties together with the happy feature that it can be built at costs not much higher than those of conventional computer memories. By contrast, Foster's price comparisons put the presently available commercial associative memory system (as of 1973) at about 200

1 Caxton Foster, Content Addressable Parallel Processors, Van Nostrand Reinhold, 1976

2 S S Yau and H S Fung, Associative Processor Architecture - A Survey, ACM Computing Surveys, Vol 9, No 1, March 1977

times as expensive as conventional computer memories.

Another way of describing the capabilities of REM is to compare it with RAM (Random Access Memory) and ROM (Read Only Memory). RAM is the conventional computer memory discussed above, called "Random Access" because it has the property that access can be had to any of its locations (even chosen at random) by contrast with serial-access memory (not further discussed in this technical note), a more limited type of location-accessed memory in which only one location is available at a time. ROM is a limited type of random access memory which has its information fixed so that it can be read from but not written into. (There is also the PROM (programmable ROM), and the EPROM (erasable PROM).) ROM, RAM, and REM are compared in the following table.

	MODES OF OPERATION			
	Read	Write	Recognize	Multi-write
ROM	+	-	-	-
RAM	+	+	-	-
REM	+	+	+	+

REM Functions

As the table indicates, REM has two functions not shared by conventional computer memories. Besides the recognition capability there is the function which Foster calls multi-write, the ability to write information into multiple storage locations in one operation. With a RAM, by contrast, it is possible to write into just one location at a time, that which is identified by the address supplied with the write instruction.

The recognition function is described above only in its simplest and most direct variety, that in which a memory location recognizes that its contents exactly match the presented data. REM also has recognition functions involving quantitative comparisons, (1) greater than or equal to, (2) less than or equal to. The three varieties of recognition are built into the REM hardware. Besides being usable directly, they provide the basis for various additional functions specifiable by software, such as (1) not equal to, (2) greater than, (3) less than, (4) between (i.e., greater than lower limit but less than upper limit).

Moreover, different functions, including no function, can be performed on different portions of REM entries. "No function" means that a certain portion of the REM entry is simply ignored; whatever happens to be there is accepted.

The use of these various capabilities is perhaps best understood with the help of an example. Let us suppose that we have a REM system loaded with a file of entries, each entry consisting of information about an individual. The entry is formatted, let us say, to consist of the following:

(1) Last name

(2) First name and initial (or initial and middle name, or . . .)

(3) Street address

(4) City

(5) State and zip code

(6) Telephone number

(7) Occupation

(8) Annual income

(9) Age

We could then, if desired, obtain a list of all the persons (1) living in Arizona, (2) no more than 35 years old, (3) with annual incomes of $20'000 or higher. Notice that different recognition functions (equal to, less than or equal to, greater than or equal to) can be performed upon different portions of the entry (including no function at all for the irrelevant por-

tions of the entry) Also, since the system is perfectly flexible as to what it does with the results of the recognition operation, we may ask it to read out just a portion of (rather than the whole of) the qualifying entries (e g , just the name and the telephone number) or we may alternatively specify that some information is to be written into some part of the qualifying entries, by means of the multiwrite operation Or, one field of the entry might contain an address to a location in a disk file where more extensive information about the individual is stored.

Note that functions such as those just described can be performed by ordinary computers; but they would be required to perform searching operations in place of recognition, and a series of individual write instructions in place of multi-write. The time of operation is considerably longer, and it increases sharply as the size of the file grows By contrast, a recognize or multi-write operation can be performed throughout REM about as fast as a read or write operation, and the time does not increase with the size of the memory.

Moreover, for elaborate specifications such as that in the above illustration, the software can get quite complex in systems with ordinary (RAM) memories And more complex software requires not only more human time for its creation, but also more memory space It is of course for just such reasons that some people have been willing to build, and others willing to buy, associative processors even at [illegible]ry high prices that have prevailed until [illegible]

Masking

A mask may be applied to any of the REM-functions, even to the ordinary location-accessed read and write operations The mask has the function of blocking out certain bits, so that they are unaffected by the operation Any pattern of 1 s and 0's can be used as a mask the bit positions for which the mask has 1 participate in the operation while those for which the mask has 0 are masked out. The size of the mask is that of the computer word, which has been set at one byte (eight bits) in the first REM systems being made available by SEMIONICS. As an example, the mask 10000000 would cause all bit positions but the leftmost to be ignored by whatever it is used with. With the multi-write operation, this mask will allow data (0 or 1) to be written in the leftmost bit position leaving the other positions unchanged Such an operation might be used to flag all records which have satisfied a preceding recognition operation.

Effective masking of byte-sized units is also provided for, but without the need for overt masks Since the Central Processing Unit (CPU) operates upon only one computer word at a time, it simply omits consideration of those which are to be effectively masked This simple practice is followed in the above illustration, in which certain entire fields (e g Last name") are ignored in specifying the recognition criteria. At smaller levels, down to the individual byte, it is just as easy to use such recognition criteria as, for example· (1) last name beginning with B (all other letters disregarded), (2) telephone area code 303, (3) first digit of the zip code greater than or equal to 7 (4) last name Anders?n (where ? indicates wild character–i e , byte to be ignored)

Complex Functions

Systems which have the capabilities described above are called Content-Addressable Parallel Processors" (CAPP's) by Foster Such machines are quite powerful By interweaving recognition and multi-write operations, with appropriate use of bit-masking, a CAPP is able to achieve speeds, flexibility, and programming ease well beyond the range of even very large and expensive computers of the conventional kind.

Thus a recognition operation, as already men-

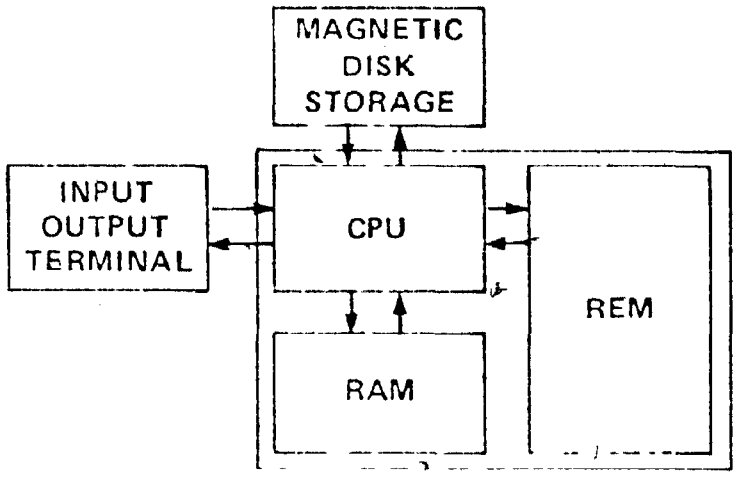
tioned, can be followed by a multi write operation, using a mask, to flag all records meeting the particular set of recognition criteria These flags may now be included in the criteria for subsequent recognition operations

It is thus easy to include either-or conditions in the recognition criteria In the above example, instead of asking for the records of all persons living in Arizona, one could specify Arizona, New Mexico, or Colorado The appropriate REM system subroutine can then multi-write a flag in all records with Arizona, then try New Mexico and multi-write a flag in the same position of these responding records, then likewise for Colorado; after which those records with the flag are the ones satisfying the disjunctive criterion

Other complex operations made possible by the abilities of REM include (1) incrementing the count field of all records meeting specified recognition criteria, (2) bit by bit comparison of an input pattern with stored patterns, (3) locating the record having the maximum value for a specified byte position or field (e g the count field), (4) like (3) for minimum value (useful in alphabetic sorting since alphabetic order corresponds to numeric in standard binary codes for alphabetic characters, (5) finding best fit in pattern recognition situations in which there is not likely to be a perfect match, by combining (1), (2) and (3) (6) printing out an ordered list (based on alphabetic or numeric order of specified field) of all records meeting specified recognition criteria, (7) moving information from one field to another within all records or all records meeting specified recognition criteria, (8) adding a constant to (or subtracting from) all records having specified properties, (9) adding two fields (or subtracting one from the other) within all records having specified properties; (10) various logical operations upon flags, etc.

The REM Data System

REM can be packaged with various types of equipment besides the CPU for different applications In the typical system, the CPU is also connected to some RAM (for programs and other information not requiring the somewhat more expensive REM) and to various peripheral devices, such as a keyboard for input, CRT and/or printer for output, and external storage on tape and or disk A typical REM Data System may be diagrammed as follows



SEMIONICS
The Claremont
Tunnel Road
Berkeley, CA 94705
(415) 548 2400

SEMICONDUCTOR
TECHNICAL NOTE

Applications and Markets for REM

Recognition Memory (REM)

In general, REM is useful wherever searching is required with ordinary computers and wherever ordinary computer software systems are being used for indexing or other means of keeping track of where data is stored. Its advantages lie mainly in greatly increased speed of processing and in simplification of software. In addition, the multi-write capability and the operations it makes available, such as parallel arithmetic, open up new vistas in computer applications which programmers and system designers will be exploring for many years to come.

Since REM can do everything that ordinary RAM can do in addition to parallel processing, and since its cost is only moderately higher than that of RAM it may become as widely used as RAM during the next 10 to 15 years. This prospect seems especially likely in view of the fact that hardware costs are continuing to decline while the cost of software, which requires programmers, continues to increase. Thus the superficially greater cost of REM systems can be repaid many times over in savings of programming costs. The potential market for REM is thus no less than vast and it appears possible that REM will revolutionize the computer industry.

No one can foresee all the new developments REM will bring after programmers and computer scientists get a chance to work with it. But several areas are immediately apparent in which a REM system has clear advantages over ordinary computers. Some of the areas are outlined below, and sophisticated computer people will have no difficulty filling in the outline with further possibilities.

1. Pattern Recognition

The basic process in pattern recognition is matching digital representations of input patterns against stored representations. REM allows such comparison to be done with the whole collection of stored representations in one operation. Other useful capabilities of REM which are helpful in pattern recognition are finding best fit among imperfect matches, masking out irrelevant or relatively unimportant bits, taking account of context restrictions, and multiple level recognition (in which first-level provisional percepts are passed on to second level of recognition, etc.).

Some specific area

- Optical character-recognition (printed text readers)
- Speech recognition (voice input to computers)
- Fingerprint identification*
- Weather pattern analysis
- Bubble chamber analysis
- Picture analysis
- Radar analysis

Wall Street Journal, 8 July 77: "San Jose's computer system in its test stage is relatively slow, taking 16 hours to compare one suspect's prints with its data base of 17,000 criminal prints. But when the system is complete, it will scan the entire file in several minutes."*

2. Information Retrieval

In its many different forms information retrieval may account for more than half of all computer usage today. The beauty of RIM for gaining access to information is that it allows the user simply to name what he wants, the record which satisfies the request recognizes, in effect, that it has been asked for. By contrast, ordinary computers can get such direct access to information only by means of the address of the location where it is stored. When they have to *find* something they must either search through multiple locations serially or use a software device like hash coding. Hash coding is quite limited in its usefulness, however, mainly because it lacks flexibility. It works only if the same search key, which gets "hashed" into an address or approximate address, is always used. For example, in a reservation system for a particular cruise line, access to a record requires (1) passenger's name, and (2) trip number. If an inquiring passenger doesn't remember his trip number he is out of luck. both items of data are used for the hashing.

In general, wherever access to records might be wanted via different search keys at different times (e.g. author or title or author-and-title in a bibliographic information system), the ordinary computer system requires a different index for each type of search key, and only those for which indexes exist can be used to retrieve information. A RIM index, by contrast, can be "multi-dimensional." For example, a single author-title catalog can be accessed by either author or title or both, or by *part* of the name of the author or title. Thus if a user doesn't know whether the spelling is MEYER or MEIER, he can ask for ME?ER.

RIM can be used in any of three different ways for information storage and retrieval depending on the size of the file. A small scale file can be stored entirely in RIM, but economics dictates the choice of a low-cost storage medium such as magnetic disk, for larger files. Intermediate-scale files can be read from such external storage one batch at a time into RIM for retrieval purposes. A large scale file, on the other hand, should typically be provided with a RIM index, containing just that information from each record that is likely to be needed for retrieval purposes. Such an index, since it is in RIM, is automatically multi-dimensional.

The following list of application areas is only suggestive:

- Automatic telephone directories
  - (1) for directory assistance systems
  - (2) eventually, for use by large subscribers
- Reservation systems
- Library catalogs
- Freight shipment monitoring
- Inventory control systems
- Customer files, marketing data
- Personnel files
- Insurance policyholder files
- Bank account and credit card information
- Medical data systems
- Computer assisted legal research

3. Linguistic Automation

Although research leading toward such goals as natural language understanding and translation has been going on for twenty years, results have been rather limited so far, for two main reasons: (1) inadequate models of linguistic structure, (2) limitations of ordinary computers, which lack recognition capabilities. Considerable progress has now been made on the first of these problems, and RIM provides the solution to the second.

The importance of the recognition capability, which is basic to language processing at all

levels (phonology, morphology syntax semantics), is perhaps best illustrated by the dictionary look-up problem In fact it is not a problem at all for RLM One has only to name the word one wants to gain access to its dictionary entry. No look-up, in the sense of searching, is required. The difference between the RLM system and the ordinary computer in this respect corresponds to that between the human being who has a knowledge of the vocabulary of a language and a person who lacks his own internal vocabulary information and must look up every word in the dictionary (a tedious process that will be recalled without much pleasure by all who have studied foreign languages) Looking up words in a dictionary requires searching but to the person who knows a word, it is simply recognized, and the recognition leads directly to the semantic and grammatical information needed to process the word

REM is likewise ideal for linguistic rule in general As a simple illustration consider a syntactic rule which says that a determiner (eg the, a' 'any') followed by a noun is a noun phrase

De No NI

Suppose that this rule is stored in RLM along with other rules and a dictionary which specifies among other things, that 'the' is a determiner (De) and 'dog' is a noun (No) Then the dog will be recognized as a noun phrase in a simple two stage recognition process After the recognition of the words 'the' and dog the sequence of syntax codes De No provided by their dictionary entries is recognized by the left-hand portion of the rule from which the identification NP can be read out or passed onto the next level of syntactic recognition

The applications for linguistic automation are many and varied, and we cannot hope to foresee all the possibilities at present One has only to consider how pervasive the use of language is in human life to get a hint of the vast scope of potential applications and products that will emerge as data processing machines are provided with more and more sophisticated linguistic capabilities

Linguistic automation as an area covers the two general processes of language understanding and language production. Most language-using devices will have capabilities in both these areas to varying degrees For example, a translating machine must have understanding capability in the source language and production capability in the target language The following list of possible devices incorporating one or both of these functions is merely suggestive

- Translating Machines
- Question-answering Devices
- The Electronic Encyclopedia
- The Automatic Telephone Operator
- Voice-Operated Data Entry Terminals
- The Voice Operated Typewriter
- Machines for Reading to the Blind
- Report Writers
- Text Editors
- Compilers for ordinary English as a programming language

4 Simple Text Processing

This area differs from Linguistic Automation in that the processor needn't have any ability to understand the texts it operates upon Rather the simple text processor performs general operations on texts without using any knowledge of the structure of whatever language the texts are written in Examples of simple text processing include those which produce word frequency counts indexes and concordances (the 'key-word-in-context' lists) Literature searches for more elaborately specified items than just words also come

continued

under this heading, examples of more complex queries include combinations of words co-occurrences of words with a specified maximum number of intervening words or characters, and parts of words (i e arbitrary character sequences), and they might also include Boolean operators Such text-searching also comes under the heading of Information Retrieval, which is further treated above.

5. Scientific Research

RLM is very effective in any type of research that involves looking for correlations in large quantities of data Some fields in which this type of investigation plays an important part with some illustrations to suggest the kind of inquiry to which RLM is well suited, are

Linguistics
Correlations of given phonological properties with geographic areas
Correlations among different features of syntactic structure

Anthropology, Sociology, Demography
Do societies or communities which have properties A, B, & C also have property D or E?

Medicine, Public Health
Correlations among specified collections of symptoms medical histories types of treatment environmental factors etc

Economics

Physics

Etc

6 Computer Science

To the computer scientist, who is involved in developing new computer applications, programming languages, and algorithms, RLM offers potentials that from the perspective of the present can only be dimly imagined Among the possibilities that can be clearly seen at present, besides those application areas treated above, RLM affords interesting opportunities for writing compilers particularly for such languages as COBOL SNOBOL and APL which are probably better suited to RLM systems than they are to ordinary computers Further possibilities that can only dimly be foreseen are likely to emerge when computer scientists have a chance to explore the implications of such capabilities as parallel arithmetic, parallel movement of data and parallel operations upon programs themselves (rather than just upon data)

7. Simulation and Modelling

This area overlaps with the preceding and with Linguistic Automation, but is distinct enough to warrant its own heading here. The two disciplines most closely involved are Artificial Intelligence and Cognitive Psychology By virtue of its essential capability, recognition The RLM system is more like the human brain than ordinary computers are It is therefore better suited to virtually all investigations in artificial intelligence and cognitive psychology which have struggled along up to now using the ordinary computer as the best available compromise despite its limitations

SEMIONICS
The Claremont
Tunnel Road
Berkeley, CA 94705
(415) 548 2400

PROGRAMMINC REM

is easy! When your REM system is operating in REM mode - that is, executing a recognize or multi-write instruction - the CPU thinks it is doing a write operation. The REM board converts it to the desired REM action For a recognize, the data written on the data bus by the CPU is a comparand - the item to be recognized. Further details are given in the RE, Programmer's Manual, a copy of which is included at no charge with each order of 1 or more REM boards

It takes only minutes to learn how to add REM operations to your repertoire if you are accustomed to programming machine language, assembly language, Basic, or what-have-you. You can use any assembler with REM, since all the CPU is doing is executing a write instruction.

To make things even easier, SEMIONICS has written a package of REM subroutines for complex REM operations, including such functions as the following (all of which operate in parallel on all REM records):

- Identify all records which have a specified character sequence

- Identify all records with zero (or all 1's) in specified byte position.

- Compare (=, $\geq$, or $\leq$) specified character string to specified field of all records, with responders flagged.

- Locate first responder (useful if there are multiple responders to a comparison)

- Erase specified byte of bit position in all records, or in all tagged records.

- Count number of bits in given byte position which match specified byte (for all records in parallel).

- Identify the record having maximum (or minimum) value in specified field. (Very useful in sorting, or for reading out responders in desired logical sequence.)

- Write specified data (1 to 255 bytes long) in all flagged records. (Very useful in sorting, of for reading out responders in desired logical sequence.)

- Increment (or decrement) specified field of all flagged records.

- Add (or subtract) specified binary number (1 or 2 bytes) to (from) specified field of all flagged records.

- Add two fields (or subtract) within all flagged records.

- Move data from one field to another within all flagged records.

- Boolean operation on flages within all records

The subroutine package for the Z-80 occupies 2K bytes of memory. An 8080 version (which requires 3K bytes) is planned for release in July, and this will be followed by an Alpha Micro version if there is sufficient demand. A higher level package for general application in information management is also in preparation.

With or without the subroutine package, you will find that programming for a REM system is far easier in most applications than for pre-REM computers. Table loop-up becomes trivial, sorting and pattern recognition become easy. You get more programming done in less time, and the programs occupy less memory space and run faster.

And you are able to do more. Information systems can be more flexible. REM makes it feasible to bring new sophistication into such areas as data base management, language data processing, and artificial intelligence.

SEMIONICS ASSOCIATES, 41 Tunnel Road, Berkeley, CA 94705 415-548-2400

[illegible]

By [illegible] S-100 bus, you convert your [illegible] Parallel Processor--the CPU [illegible] just as a RAM board (and it [illegible] bly--with its RAM functions) it is [illegible]:

- [illegible]
- Single voltage [illegible] nt
- [illegible] time
- [illegible]
- [illegible], including solder paths on [illegible], plated through holes, etc.
- [illegible] with large RAM systems (not needed [illegible] in parallel!)

But it [illegible] ordinary microcomputer to a Content [illegible] sor (C/PP):

[illegible]:

[illegible] to	not-equal to
[illegible] to	greater than
[illegible]	less than

Read [illegible]
Multi-write [illegible], all responders, or all non-responders)
[illegible]

Recognize and multi-write are parallel operations: executed for all RAM records with a single instruction. Execution time is 4 microseconds, no matter how many RAM boards are [illegible]--it does not increase with increased memory size. Also, since [illegible] processed in parallel anyway, multiple RAM boards [illegible] space. The mask--any desired combination of 8 bits [illegible] with recognize or multi-write, but also with ordinary [illegible] bits in positions where the mask has Ø are [illegible].

Specifications:

[illegible]	[illegible]
[illegible]	[illegible] bit words, 256 word RAM records ("super-words"
[illegible]	Static, CMOS, single voltage
[illegible]:	[illegible] nanoseconds
[illegible] Cycle [illegible]:	[illegible] nanoseconds
[illegible]:	=, [illegible], >, <
[illegible] Time:	4 microseconds
[illegible] Time:	4 microseconds
Bus:	S-100
[illegible]:	+ 8 volts @ 2.5
[illegible]	0-55° C

Price: [illegible]

Screensplitter

from Micro Diversions

Roberts Information Services, Inc. (ROBINS)
8305-G Merrifield Ave
Fairfax, Virginia 22030
Phone (703) 560-7888

INTRODUCTION

SCREENSPLITTER is a complete text-oriented TV display system for S-100 compatible hobbyist, small business and OEM microcomputer systems. SCREENSPLITTER represents a significant advance over other TV text display systems in three important respects:

* The quantity of displayed text is large. SCREENSPLITTER displays 40 86-character lines of upper-lower case text on a standard (10 mhz or better) TV monitor;

* Onboard software is provided. One kilobyte of display driver software, the Window Package, is provided in the basic system directly on the board in a 1K 2708 EPROM. This software defines 20 user-callable functions that permit the definition and control of any number of input/output "windows" on the display screen;

* The character generator is user-reprogrammable. Since SCREENSPLITTER's character generator is a 2708 EPROM, the user may design and implement his own character set.

The SCREENSPLITTER hardware is mounted on a single, high quality S-100 compatible PC board. The kit comes complete with sockets for all 48 IC's, all discrete components, and 8 feet of coaxial cable with connector.

SCREENSPLITTER occupies one slot of the host S-100 buss, requiring approximately 1.5 amps at +8 volts, 100 ma at +18 volts, and 100 ma at -18 volts (all unregulated). Virtually all 7400 series logic is low power Schottky, and the board presents no more than one standard TTL load to the host buss. SCREENSPLITTER's output data buss lines are capable of driving 20 standard TTL loads. Table 1 lists all S-100 lines used by SCREENSPLITTER.

Physical

The text display buffer is 4K bytes of static RAM (8-2114's). Onboard software resides in a 2708 EPROM, with another 2708 EPROM serving as the character generator. The 1K byte software 2708 may be upgraded to a 2716 (for 2K of onboard software) via jumper reconfiguration.

TV sync is generated by the industry standard MM5320 sync generator, using an onboard 10 mhz crystal time base. This, in conjunction with the use of synchronous counters in the display logic, results in a very stable and crisp display.

For extensibility and compatibility with other TV display systems SCREENSPLITTER makes the 14 signals shown in Table 2 available to the outside via a 16 pin DIP socket at the top of the PC board.

Table 1.

S-100 buss lines used by SCREENSPLITTER.

Mnemonic	Function	S-100 Pin
A15	address line 15	32
A14	address line 14	86
A13	address line 13	85
A12	address line 12	33
A11	address line 11	87
A10	address line 10	37
A9	address line 9	34
A8	address line 8	84
A7	address line 7	83
A6	address line 6	82
A5	address line 5	29
A4	address line 4	30
A3	address line 3	31
A2	address line 2	81
A1	address line 1	80
A0	address line 0	79
DI7	data-in line 7	43
DI6	data-in line 6	93
DI5	data-in line 5	92
DI4	data-in line 4	91
DI3	data-in line 3	42
DI2	data-in line 2	41
DI1	data-in line 1	94
DI0	data-in line 0	95
DO7	data-out line 7	90
DO6	data-out line 6	40
DO5	data-out line 5	39
DO4	data-out line 4	38
DO3	data-out line 3	89
DO2	data-out line 2	88
DO1	data-out line 1	35
DO0	data-out line 1	36
SMEMR	status, memory read	47
MWRITE	memory write	68
PDBIN	power data buss in	78
PSYNC	power sync	76
PHI-2	second phase clock	24
SWO-NOT	status, memw or outp	97
SOUT	status, outp	45
PRDY	processor ready	72
+8	power supply	1,51
+18	power supply	2
-18	power supply	52
GND	power supply	50,100

Table 2.

External signals generated by SCREENSPLITTER.

PIN 1	10 mhz clock
PIN 2	10 mhz clock, inverted
PIN 3	Field
PIN 4	Composite Sync, inverted
PIN 5	Horizontal Blanking
PIN 6	Horizontal Blanking, inverted
PIN 7	Vertical Blanking
PIN 8	Vertical Blanking, inverted
PIN 9	Final Video, TTL level (less sync)
PIN 10	Composite Video, 2 volts p-p
PIN 11	Character Generator, bit 7
PIN 12	Character Generator, bit 8
PIN 13	Character Winver
PIN 14	Video Suppressor

SCREENSPLITTER is designed to be located in the host system's address space on an even 8K boundary, as shown in Figure 1 Address lines A15 A14 and A13 are jumper selectable to allow user selection of the 8K boundary.

The display buffer appears to the host CPU as a 4K block of 450 ns (or faster) static RAM. Organization of the display buffer is as a 40 by 88 byte array. The first 86 bytes of each line correspond to visible screen characters, with the 87th byte unusable, and the 88th byte available to the user for internal line identification, etc.

To write a character onto the screen, the host system stores an ASCII code in the appropriate array cell. Hence, although the Window Package is ordinarily used for all display control, the 4K display buffer is directly accessible to the host system. If desired, the host can actually use SCREENSPLITTER's 4K static RAM display buffer for computing during periods for which a meaningful display is not required.

Logical

The Window Package software appears to the host CPU as a 1K block of 450 ns EPROM, situated in the first 1K of the 4K block of memory immediately below the display buffer in the host's address space. This 1K is fully decoded, so that the user can map 3K bytes of his own RAM in the unused portion of the lower 4K block. Users who prefer to situate the Window Package elsewhere in memory can do so (after software relocation) by disabling the onboard 2708 EPROM. In this case, SCREENSPLITTER occupies only 4K of the host system's address space (still on an even 8K boundary).

Because the entire display buffer is mapped into the host's address space, relatively high speed display transactions can occur (see the Product Specifications). SCREENSPLITTER's circuitry is designed to permit the host CPU to run at full speed with no wait states, and has logic for suppressing the white-on-black "snow" often associated with such unimpeded buffer access. A jumper permits optional introduction of one wait state for memory accesses, making SCREENSPLITTER fully compatible with 2 and 4 mhz Z80-based systems. as well as 8080-based systems.

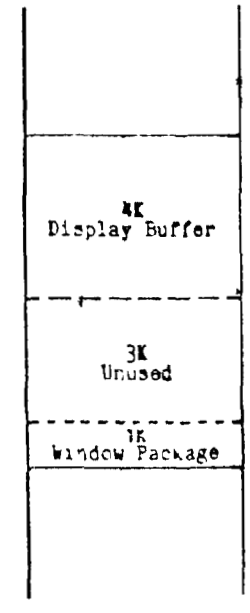
SOFTWARE

Physical

SCREENSPLITTER's Window Package software module is supplied in a preprogrammed 2708 EPROM, origined at any requested 8K boundary. Since the Window Package 2708 is jumper-upgradable to a 2K byte 2716, the user can extend the onboard software by another 1K bytes. (Micro Diversions will be releasing a 1K byte page-oriented text editor which commands the powerful Window Package functions in the basic 1K module.)

Full object code (with symbol table), and thoroughly commented source code listings for the 1024 byte Window Package, as well as the Window Package User's Manual and Applications Notes, are provided with both kits and assembled units. Additionally, hex listings and pictorial plots of the character generator EPROM's are provided to facilitate user development of custom character sets.

Figure 1.

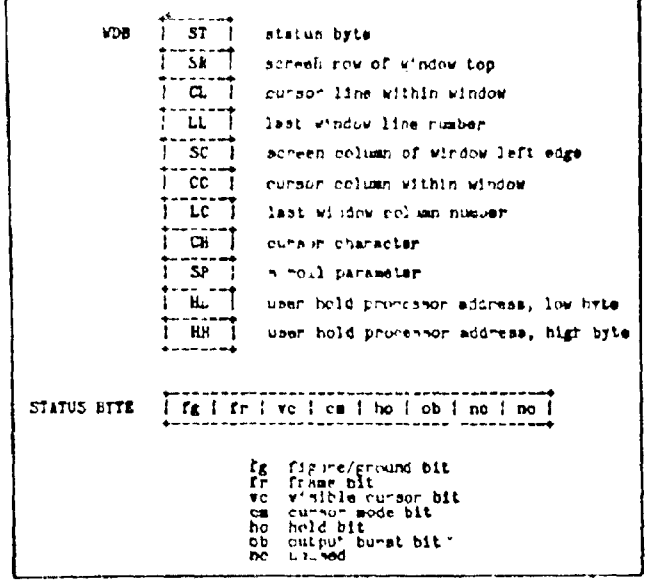


Memory map of SCREENSPLITTER in host address space.

The onboard Window Package defines 20 user-callable functions for controlling and writing to the display buffer (see Table 3). The basic unit of control is the window. Virtually any number of windows (rectangular subregions of the screen) can be defined and independently controlled. All status information for each window is maintained in the window's window descriptor block (WDB), an 11 byte block of RAM supplied by the user at the time the window is opened (figure 2). In all transactions with window functions, the user loads the address of some WDB into the HL register, loads any additional parameters into registers B, C, D and E, then CALLs the desired window function. Window Package functions generally destroy all registers.

With the exception of two references to the display buffer's location in the host address space, the Window Package is reentrant, and requires less than 64 bytes of system stack to run (A special, fully reentrant version of the Window Package that controls multiple SCREENSPLITTER displays from a single 1K EPROM is available.)

Figure 2 Window descriptor block format.



Windows

Each window defines a rectangular region of the display screen from size 1 by 1 up to 40 by 86. Each window can be manipulated independently from all other windows. Overlapping windows are permitted, but operations on a window take place without regard for possible effects on any overlapping windows.

Each window has its own set of parameters governing:

1. whether the window's figure/ground is reversed (black text on white background
2. whether the window has a frame (visible border)
3. whether the window's cursor is visible
4. what character is to be used as the window's cursor
5. whether the cursor displaying technique is to print the cursor character at the cursor location, or simply to reverse the figure/ground of the character at the cursor location
6. whether the window is to be held (by calling a user-specified "hold processor") at scroll time during output bursts
7. the type and degree of scrolling (pop-up or wraparound) the window will perform at window-full time.

Window Package Functions

All the following functions perform error checking; detection of an erroneous request occurs before any alterations to window descriptor blocks or the visible display are made.

INIT(ch)
Clears the entire display buffer to character CH (typically ASCII blank).

OPEN(wdb,x,dx,y,dy)
Opens a window of size DX rows by DY columns with top-left corner at screen row X, screen column Y; initializes the window's window descriptor block, WDB; does not clear the region.

CLEAR(wdb)
Clears the window and resets the cursor to the top left.

FRAME(wdb,hc,vc,cc)
Frames the window, using HC for the top and bottom borders, VC for the left and right borders and CC for the four corners; reduces the window's interior region by two characters in each dimension, and clears the window.

UNFRAME(wdb)
Removes the window's frame (if any); if removed, increases the window's interior by two characters in each dimension and clears the window.

LABEL(wdb,str,len)
If the window has a frame, prints the string pointed to by STR, of length LEN, centered on the window's top border.

LABELS(wdb,str)
Behaves identically to LABEL, except that the string is terminated by the string terminator character (octal 377), so that a length parameter is not required.

CURSORCH(wdb,ch)
Defines CH to be the window's cursor character; automatically complements CH if window's figure/ground is reversed.

SCROLL(wbd,n)
Sets the window's scroll parameter to N (0 for wraparound greater than zero for N-line pop-up).

PRINT(wdb,str,len)
Prints the string pointed to by STR, of length LEN, to the window, starting at the current cursor position; LEN may be from 0 (no characters printed) to 65,535; the cursor is forced to the beginning of a fresh line after the print.

PRIN(wdb,str,len)
Behaves identically to PRINT, except that the cursor is not forced to a fresh line after the print.

PRINTS(wdb,str)
Behaves identically to PRINT, except that the string is terminated by the string terminator character (octal 377), so that a length parameter is not required.

PRINS(wdb,str)
Behaves identically to PRIN, except that the string is terminated by the string terminator character (octal 377), so that a length parameter is not required.

FRESHLINE(wdb)
Forces the window's cursor to the beginning of a fresh line if it is not already there.

CLEARLINE(wdb)
Clears the window's cursor line, resetting the cursor to the first character of the line.

BACKSPACE(wdb)
Backs the window's cursor up one character if not already at the window's leftmost column; erases one character as it does so.

COMPLEMENT(wdb)
Reverses the figure/ground of the window's interior.

FLASH(wdb)
Reverses the figure/ground of the window's interior momentarily (about a third of a second for 2 mhz CPU).

PLOT(wdb,x,y,ch)
Prints character CH at window line X, column Y; does not affect the cursor position, and cannot invoke scrolling.

MOVEWINDOW(wdb,x,y,ch)
Moves the window so that the window's top left corner is on screen row X, screen column Y; fills any vacated region of the screen with character CH (typically ASCII blank).

Table 3 Window Package function summary.

Mnemonic	Relative Entry Address (octal)	Registers B	C	D	E	HL
INIT	31	ch				
OPEN	50	x	dx	y	dy	wdb
CLEAR	255					wdb
FRAME	753	hc	vc	cc		wdb
UNFRAME	1152					wdb
LABEL	1215		str		len	wdb
LABELS	1212		str			wdb
CURSORCH	1316	ch				wdb
SCROLL	743	n				wdb
PRINT	632		str		len	wdb
PRIN	626		str		len	wdb
PRINTS	617		str			wdb
PRINS	610		str			wdb
FRESHLINE	1516					wdb
CLEARLINE	311					wdb
BACKSPACE	1473					wdb
COMPLEMENT	1333					wdb
FLASH	1412					wdb
PLOT	1443	x	y	ch		wdb
MOVEWINDOW	1545	x	y	ch		wdb

CHARACTER GENERATOR

SCREENSPLITTER's character generator represents characters as 5 by 8 pixel grids. Two standard character sets (Graphics and Scientific) are available, each defining 32 useful symbols in the otherwise non-printing first 32 ASCII codes. An APL character generator is available optionally.

The standard font is an aesthetic upper/lower case design, with underhang for the lower case characters which require it. Any character can be defined as an inherently winking character by programming high bit 6 of each of the character's scan lines in the 2708 character generator. In fact, since each scan line has its own wink bit, it is possible to generate partially winking characters. (There are two in the Graphics character set.) Bits 7 and 8 of each scan line in the character generator, not used by SCREENSPLITTER, are made available externally for user extensions (Table 2).

The high-order bit of each byte in the 4K TV display buffer is interpreted by the display logic as the character's figure/ground. Thus, each of the 3440 visible screen characters' figure/ground is independently controllable.

EXAMPLE APPLICATIONS

1. As an interface to a high level language, such as BASIC

   In this application, each important subroutine and function is assigned its own private window. As it runs, each subroutine or function can output trace information, status indications, or user prompts through its own window. The visual effect would be flurries of activity from window to window, each independently scrolling, flashing, etc. This provides a very effective way to see "in two dimensions" exactly what's going on inside your programs. User typein can be directed through individual windows, giving the illusion of making it possible for the user to converse with components of a large system independently.

2. As a Debugging Display System

   In this application relevant status information concerning the execution of a program (as output by a debugging package) is displayed through numerous windows. One window, for example, could display the register and accumulator contents; another could display the top N items on the system call stack; another could flash up interrupts as they were serviced; another could display a selected portion of central memory, such as a critical array; another could display a real-time/run-time clock. You could even do all these things independently for a number of subroutines, presenting and recording, say, the registers and accumulator as they were at subroutine exit time. You could also arrange to have the normal I/O of your program appear in one window, with debugging information popping up through another window beside it when requested.

3. As a Basis for Controlling Several Keyboards

   In this application, there are several keyboards, each with its own window, as might be useful in multiple player computer games. Each player would have his own area on the screen, into which all input typed by him would be echoed, and to which all output directed at him would be written. Under certain circumstances, a player might be given access to another player's window or "party" windows might be established to combine the inputs from several users into a single window.

4. As a Basis for Advanced Page-Oriented Text Editing

   In this application, entire pages of text are displayed on the screen in one relatively large window As you decide to move paragraphs or lines around, you issue commands to pick up a paragraph or line and place it in a smaller holding window. As you rummage through the main window looking for the spot at which to insert the line or paragraph, the holding window remains fixed. Finally, you issue a command to insert the contents of the holding window at a selected point in the main one. The editor would perhaps be capable of directing its editing powers at the text in any window, so that you could also modify the line or paragraph in the holding window before reinserting it into the main window. Also, using the MOVEWINDOW function, you could actually lay out a screen of text (as it is about to be printed on a hard-copy device) by moving paragraph-size chunks of text around. Meanwhile, of course there could be a very small window up in the corner containing a real time clock ticking away!

5. As a Basis for Networking and Concurrent Processing

   In this application, you might wish to be doing local computing, but remain connected to some external computer (over the phone lines) or to a computer network of other personal machines similar to yours. One window would then be reserved for all I/O in your local computation, with additional windows through which communication with the external computers could occur. You might allocate one window for each other personal computer attached to the network. This would enable you to keep all the I/O to the various communicating computers separate from each other, and separate from your local computations. Again, party windows could be established.

PRODUCT TECHNICAL SPECIFICATIONS

Mechanical	Nominal board size	5.0" x 10.0" (less edge card fingers)
	Nominal board height (with components)	0.75"
	Coaxial cable type	RG-59
	Coaxial cable length	8'
	Coaxial cable connector style	PL-259
	Number of IC's	48
	IC socket style	low profile, solder
	Board construction	G-10 material, solder mask both sides
Electrical	Power requirements	1.5 amps @ +8 volts (unregulated) 150 ma @ +18 volts (unregulated) 100 ma @ -18 volts (unregulated)
	On-board regulation capacity	2 amps @ +5 volts 1 amp @ +12 volts 1 amp @ -5 volts 50 ma @ -12 volts
	On-board power supply filtration	100 microfarad electrolytic (+8 volts 6.8 microfarad tantalum (+8, +18, -18 1.0 microfarad tantalum (+5, +12, -5) 0.01 microfarad disc (+5 despiking)
	IC types	2114 4K static RAM (display buffer) MM5320 sync generator 2708 EPROM 7400 Series Low Power Schottky 7400 Series Standard Buffers
	Loading of host buss logic lines	no more than 1 standard TTL load
	Buss driving capability, data output lines	20 standard TTL loads
Timing and Logical	Read, write cycle time, display RAM	500 ns (nominal)
	Read cycle time, Window Package Software	500 ns (nominal)
	Number of user-selectable wait states	0, 1
	Location in host address space	any 8K boundary (jumper selectable)
Video	Visible display size	40 lines, 86 columns
	Minimum Frequency rating; TV monitor	10 mhz
	Recommended monitor size	13" or larger (9" are acceptable)
	Recommended monitor phosphor, low ambient lighting conditions	Normal (fast decay)
	Recommended monitor phosphor, high ambient lighting conditions	P-39 or similar (slow decay)
	Output level, composite video	2 volts peak-to-peak (nominal)
	Output impedance	75 ohms (nominal)
	Writing character rate	2/sec (approx.)
	Character grid size	5 by 8 pixels
	Inter-column spacing	1 pixel
	Inter-row spacing	4 scan lines
	Figure/ground control	high-order bit of each display buffer byte
Software (Logical)	Size	1024 bytes
	Number of user-callable functions	20
	Assembled location	any 8K boundary (user-specified)
	Number of logical windows	(no limit - windows may overlap!)
	Format of supplied object listings	both octal and hex
	Window scrolling options	wraparound and N-line pop-up
	Window controls	figure/ground, cursor, frame, movement, output burst hold
Software (Timing)	(All times are for a 2mhz host CPU; all times are approximate.)	
	Full screen clear	0.06 sec
	Full screen complement	0.1 sec
	Window flash (delay loop)	0.4 sec
	Basic print rate to a window	2300 characters/sec
	Window move (20 by 40 window)	less than 0.03 sec

American Journal of Computational Linguistics Microfiche 76: 62

The TARGET Project's Interactive Computerized Multilingual Dictionary

John Burge
Departments of Modern Languages and Computer Science
Carnegie -Mellon University, Pittsburgh, Pa. 15213

Summary:

This document is a brief introduction to Carnegie-Mellon University's interactive computerized multilingual dictionary. It describes the use of this dictionary both by *translators* in the course of their work and by the *terminologists* responsible for updating and maintaining it. This discussion is placed in the context of the overall effort (known as the *Target Project*) to provide aids to translators. A final section presents the solution to the problem of representation of term equivalence adopted in Target.

1. The Target Project

Target is an interdisciplinary research project undertaken jointly by the Translation Center and the Department of Computer Science at Carnegie-Mellon University to investigate and develop computer aids for language translation.

Since high quality automatic translation does not seem to be immediately realizable, our efforts at introducing computerization into the translation task have been directed towards providing practical aids for translators. Working with the assumption that each translator can be provided with a standard video terminal connected by a dial-up line to a remote computing facility,[2] we are exploring primarily two aids. They are (1) an interactive multilingual dictionary, and (2) an environment consisting of (1) plus text manipulation facilities within a windowed page editing environment. The latter research will be described in a future AJCL paper; this document justifies and describes only the former, how it is accessed and how it is built up and maintained.

---

[1] It is an amended version of an informal description of the TARGET and TERMIN programs demonstrated at the Foreign Broadcast Information Services seminar on Aids to Translators, Washington, DC, May 1978

The configuration in daily use by the Translation Center at Carnegie-Mellon University involves a Lear Siegler ADM-3 terminal connected by a 300 baud dial up line to a PDP-10 run under the TOPS-10 operating system by the Computer Science Department and shared simultaneously by users working on many different projects

The primary motivation for the interactive dictionary is that a technical translator may spend up to 60% of his or her time simply looking up terms. This may include unsuccessful searches in several dictionaries, partly because these dictionaries are out of date by the time they are published. An interactive computerized dictionary would provide effectively "immediate" access to entries and moreover could be kept constantly updated at the central computing facility.

Currently the dictionary contains specialized terminology in English, French and German in a number of fields Specialized terminology was chosen because this is often most helpful in practice to the professional translator and also because this is where the benefits of standardization could be most immediately apparent The languages were chosen because they are the most immediately useful in the local environment, as were the fields (mainly finance, business and iron, steel and mining technology).

The next section shows in some detail how a translator would access the dictionary and determine a correct equivalent The section after that describes the facilities used to maintain and augment the dictionary The interface to the dictionary described in the next two sections represents the fruits of continual close cooperation over an extended period of time between researchers from the Computer Science Department and from the Department of Modern Languages Such cooperation, while it presents many problems initially, is a *sine qua non* of success in a venture such as Target.

While performing initial studies for the representation of equivalence between terms the most central relation in a multilingual dictionary -- we have departed from the common practice of using an alingual set of concepts realized differently in different languages Close examination showed that this could not accommodate some nuances of meaning in disparate languages and was not precise enough for making inferences when a particular equivalence was not already present in the dictionary. Moreover, it was found to be less efficient than another method which was investigated and ultimately adopted. Some arguments proposed for adopting this different method are set forth in the final section of this document

2. The TARGET Program

TARGET is also the name of the program used by translators to access the entries in the dictionary while doing their translation work This section describes how it is used. The illustrations are exact traces of the interaction between the program (in a roman font) and the translator (in an italic font).

We are first asked for the term names and the languages we wish to translate From (*i.e.* the source language) and To (*i.e.* the target language):

```
Term: bond
From Language: en
To Language: fr
```

Now, if there is only *one* equivalent for that term between those languages, we shall get that equivalent directly. In this case we have a choice to make:

```
  Term: bond
  From Language: en
  To Language: fr
bond    Chemistry: Theoretical Chemistry;   (ch4)
        The Nuclear Industry: Nuclear Energy;   (at6)
        Financial Affairs - Taxation - Customs;   (fi)
  Select Code:
```

Let us say the article we are translating is in Chemistry. Then we just type the appropiate code. These are in parentheses in the example and are the same codes as used in the EEC's Eurodicautom system. Here we select a code:

```
  Term: bond
  From Language: en
  To Language: fr
bond    Chemistry: Theoretical Chemistry;   (ch4)
        The Nuclear Industry: Nuclear Energy;   (at6)
        Financial Affairs - Taxation - Customs;   (fi)
  Select Code: ch4
```

and we shall get the appropriate fiche:

```
  Select Code: ch4

bond    liaison (FR)

        Chemistry: Theoretical Chemistry;
        The Nuclear Industry: Nuclear Energy;

Reference Terms:    bonding energy

  Term:
```

We have been told that the same equivalent is used for both chemical, and nuclear bonding. Had there been further information, such as a usage sample, a definition or a note, we would have been asked whether we wanted to see it with the question *More?* Answering *yes* would show the information to us.

After this first use, Target assumes that we are translating from English to French Notice that it does not ask us the From and To questions:

```
  Term: bond
bond    Chemistry: Theoretical Chemistry;   (ch4)
        The Nuclear Industry: Hazards;   (at8)
        Financial Affairs - Taxation - Customs;   (fi)
  Select Code: fi

bond    emprunt (FR)

        Financial Affairs - Taxation - Customs;

Reference Terms:    government bond
  Term:
```

We can override the assumption by typing *all on one line* the term name, the source language and the target language. Here we check on the equivalent just obtained:

```
Term: emprunt fr en

emprunt    bond (EN)

        Financial Affairs - Taxation - Customs;

Term:
```

English and French now become the new anticipated source and target languages, respectively.

3. The TERMIN Program

TERMIN is the name of the program used by *terminologists* to augment and maintain the dictionaries. Clearly the facilities in TERMIN must be more varied than the simple retrieval facility which is TARGET; the facilities in TERMIN are accessed through a set of *commands*:

```
Help                    (to get instruction)
Create Entry            (to enter a term)
Exit                    (to leave TERMIN)
Retrieve Entry          (to get term and term information)
Edit Entry              (to revise or augment entry)
Delete Entry            (to delete an entry)
List Contents           (to list terms in one language)
Target                  (to get target language equivalent)
Record Transaction      (to record session)
Term Hardcopy           (to make hardcopy of a term)
Dictionary Hardcopy     (to make hardcopy of a whole dictionary)
Regenerate              (to correct faulty dictionary)
Recover Space           (to recover space used by deleted and updated entries)
Do                      (to execute a command file)
Start                   (to start using an Option)
Stop                    (to stop using an Option)
```

The TARGET program essentially just repeats the TERMIN command of the same name.

The commands are the large functional units in terms of which we interact with the dictionary. We use them when entering new terms (*Create Entry*), editing existing terms (*Edit Entry*), printing dictionaries[3] (*Dictionary Hardcopy*), etc. -- and in fact for everything we do with the dictionary. A short description of the use of each command follows.

---

[3] An example page is reproduced in an Appendix, exactly as it was printed by our Xerox Graphics Printer.

3 1 Help

This provides on-line access to written comments on various aspects of the use of the TERMIN program by terminologists. All of the twenty or so texts which may be accessed in this way were written by the terminologists (who also ordered the list of commands as above and provided the brief description beside each one).

```
>help
    Help on: altering
ALTERING is a way to correct errors without typing the whole line
again.  You can use it by using the Option ALTERING before you use
the Command EDIT:
        >start altering
        >edit
ALTERING will continue until you exit from the program.  If you want
to go back to the regular way of editing before then, do
        >stop altering
This is a summary of ALTER mode commands:
        <SPACE BAR> : advance the cursor by one character
        control-H: back up one character (same as rubout or BS)
        D : Delete the next character
        J : Join this line with the next, ie. delete the next
                carriage-return in the text
        H : Type this help text
        L : List - type the whole string
        Q : Quit editing the string and ignore all the changes made so
            far
        P : Print string, putting the cursor back to the same position
        T : Transpose the next character with the one after it
        V : Invert the case of all the characters in the current
                word, starting with the next one
         ESC> : escape from insert mode (=<ALT>)
        ^ = <RETURN  : terminate editing of the string
        Ax : Add the next character to be typed, ie. 'x' here
        Cx : Change the next character into 'x'
        Fx : Kill (ie. delete) all characters until the next 'x'
        Sx : Skip to the next occurrence of character 'x'
        I : Insert all characters typed in, starting at the current
                position, until an <ESC> is pressed
        Mx : Munch up characters till the next 'x', then go into
                insert mode
        R : Replace - delete the next character and then start
                inserting.
        X : Extend - go the end of the string and start inserting
```

Most of the texts provided in this way give hints and reminders in an informal manner, rather than a detailed sequence of instructions on how to use the command.

3 2 Create Entry

This command is used to create new entries in the dictionary for terms which have not yet been entered Let us suppose that we have prepared a fiche specifying the equivalent in French of the English term *bond* appropriate for the field of Chemistry. Space constraints prevent a detailed description of the interactions leading to the entry of this term into the dictionary, but the following is a trace of the process:

```
>create
  Term: bond
  Language: en
  Field Classification: ch4
  Equivalents: liaison
    Language: fr
  Grammatical Categories: nou
Select  Usage Sample
        Definition
        Reference Terms
        Note
        Synonyms? reference
  Reference Terms: "bonding energy"

bond    liaison (FR)
        CH4

Reference Terms:    bonding energy

nou

! New term "liaison" being entered in "FR"%
```

TERMIN does what it can to save us effort; here it has generated a small fiche to contain each equivalent which does not already exist. The minimal fiche it generated for *liaison* is this:

```
liaison   bond (EN)

        Chemistry: Theoretical Chemistry;
```

This will usually need to be augmented with other information, such as the grammatical category This is done by using the *Edit Entry* command (see below).

3 3 Exit

This is how we leave TERMIN:

```
>exit

EXIT
```

The user is now no longer using the TERMIN program, but is using the TOPS-10 monitor.

3.4 Retrieve Entry

This command is used to print whole entries at our terminal. Here is *bond*:

```
>retrieve
  Term: bond
  Language: en

bond     liaison (FR)
         Chemistry: Theoretical Chemistry;

Reference Terms:     bonding energy

nou
```

There is only one Equivalent, one Field Code and one Reference Term for *bond* at this point. More complex entries, with more of the optional term information, will display all that information as well. *Retrieve Entry* shows *all* the information there is for *bond*; only portions may be displayed with the *Target* and *Edit Entry* commands.

3 5 Edit Entry

There is a number of reasons why it may be necessary to edit an entry. Perhaps the person who entered it mad a typo, perhaps it is necessary to extend an entry to include, say, a Usage Sample (or perhaps to replace the old one with a better one) or perhaps a new fiche must be entered for an existing term.

Update the Field Code: Suppose that further investigation of the term *bond* has revealed that the same French equivalent is also used for the "bond" holding the nucleus of atoms together as for the electronic bond which keeps different atoms

together. Had this been known when the original fiche was created, it would have contained *two* field codes, *ch4* and *at8*. We need to update the original entry.

Add a New Fiche: The term *bond* is also used in financial and commercial circles (among others[4]), but here the equivalents in French are *not* liaison So we can enter a new fiche for this term.

Deleting a Fiche: It is occasionally necessary to delete a fiche within a term, but not the whole term.

Edit Entry is used to accomplish all these functions. It is the major tool used in maintaining the dictionaries.

3 6 Delete Entry

This is used to delete entire entries:

```
>delete
  Term: "blast off valve"
  Language: en
>
```

It will no longer be accessible

3 7 List Contents

This gives an alphabetical list of terms in a specified language. We can limit the list by specifiying the first one or two letters of the first and last term. It may be aborted by typing @ at any time, and will return us to the command level immediately.

Let us get a list of the English terms from *bo* through *c*:

```
list
 Language: en
 From Letter: bo
 To Letter: c
        'body centered cubic
        'bold'
        'bond'
        'bonding energy'
        'boom'
        'bore'
        'boring bar'
        'boundary position'
        'bracket'
```

---

[4] Note that there is nothing to stop different fiches for the same term sharing field codes A different fiche is needed whenever, and only when, the equivalents are distinct

'breakdown'
'breaking'
'bridge connection'
'broker'
'brokerage'
'buffer'
'burden'
'business'
'business profit'
'butterfly valve'
'by means of'
'by-pass valve'
'by x-ray diffraction'
'capacity'
'capital'
'capital and reserves'
'capital gain'
'capital goods'
'capital-intensive'

3 8 Target

Target is the command (or program) we use to find equivalents for a term The TERMIN command *Target* works identically to the *TARGET* program (described above) with the exception that in TARGET if an abort character @ is typed in answer to the *Term:* prompt, the progam exits; in TERMIN, an @ at this point gets us back to the command prompt >

3 9 Record Transaction

This command is used to keep a record of the interaction between the program and the user It is used for studying how users interact with the system in order that it may become better tailored to their needs (All of the examples in this document were drawn directly from records made in exactly this manner). Each interaction between the system and the user may also be timed in the record by using the *Timing* option. This provides an extra tool for studying how the system is used in practice. It is also useful for some purposes to be able to annotate a record while it is being produced. TERMIN will ignore any line beginning with a semi-colon:

>; *This shows that comments get into the record*

3 10 Term Hardcopy

This command is used to print a specific term in the same format as that of the terms in the Appendix

The file just generated contains formatting information as well as the term itself. It must be *compiled* by the PUB document formatter

3 11 Dictionary Hardcopy

This is like *Term Hardcopy*, but the program will select the terms for us and put them in alphabetical order. We can choose the language, the initial two letters of the first and last terms, and can also restrict the fields for the terms, by specifying a set of field codes -This allows us to make selective microglossaries, choosing perhaps just those terms relevant to to the Petroleum Industry or Medicine. Here we illustrate obtaining an entire dictionary One page of it is reproduced as an Appendix.

```
dictionary
 Source Language: fr
 From Letter: a
 To Letter: z
 Restricted Fields? no
.....................................................................
.....................
Please see TEMP:FRENCH.PUB
```

This file must be formatted with the PUB document compiler. The title page of the hardcopy will describe any limits we may have imposed on its contents.

3 12 Regenerate

This command is used to re-establish the links between the various files which contain the dictionary. They can become incorrect when the computer crashes while certain operations are being performed, or when there are problems with the system.

3 13 Recover Space

When a term is stored after having been *Edited*, a new entry is made for the new version of the term The old entry is, however, still there, and hence takes up space. The same is true in the case of the *Delete* command. When a term is deleted, it actually only becomes inaccessible -- and so it is taking up space. Every so often this "wasted" space (which actually provides the potential for some backup) is recovered using this command; it compacts the dictionary

3.14 Use of Commands

As can be seen from the foregoing, to use a command we type its name (e.g. *help*). Upper case and lower case are equally acceptable. The program will then begin to prompt us for the further specifications neccessary to carry out the command. We may *Type Ahead* the responses to these questions, in which case the prompt is not given Help is usually obtainable by typing ? and any command can be aborted by typing @ in response to any prompt

3.15 Conveniences of Interactions with User

One convenience in TERMIN is that it is often not necessary to type the whole of the response to a prompt In fact, all we need to type is an unambiguous abbreviation, thus:

```
retr
```

for *Retrieve Entry*, for instance. If we type an ambiguous abbrevation, the system can help us out:

```
>re

? re is ambiguous:
        Retrieve Entry          (to get term and term information)
        Record Transaction      (to record session)
        Regenerate              (to correct faulty dictionary)
        Recover Space           (to recover space from deleted terms)
>reco

? reco is ambiguous:
        Record Transaction      (to record session)
        Recover Space           (to recover space from deleted terms)
```

We can also be assisted when we make typing mistakes:

```
>lslt
   ... did you mean LIST CONTENTS                (TO LIST TERMS IN ONE LANGUAGE) ?yes
```

As it happens, users of the system often find this kind of help confusing initially and so there is an *Option*, called *Helpful*, to control it. Initially this option is turned off, but it can be turned on simply by typing:

```
start helpful
```

TERMIN can often anticipate the answer to one of its questions. For instance, if we

have just *Retrieved* a term and then we issue an *Edit* command, the chances are that the term we just retrieved is the term we want to edit. The *Defaulting* Option makes similar assumptions To use this option, we type:

```
>start defaulting
```

and then we can utilize it:

```
>retr liaison fr

liaison   bond (EN)

        The Nuclear Industry: Nuclear Energy;
        Chemistry: Theoretical Chemistry;

Reference Terms:      l'energie de liaison

nou mas

>edit
  Term: [liaison]
  Language: [FR]
```

(Note that we have accepted the default by simply striking the <RETURN> key.)

When accessing terms, we need not specify accents (unless they distinguish between two terms). This is a convenience to be used when *accessing* fiches and their contents: when we type the *text* in a fiche we must use the correct cases and accents.

4. Some Theoretical Issues

While designing the representation of terms and their interconnections within the dictionaries, researchers at the Target Project discovered some difficulties with some of the methods adopted by other terminology banks. This section is a brief presentation of some of them.

When a sense in one language is translated by a sense in another, they are said to be *equivalents* of each other This is what is crucial to the translation task, and what is under discussion in this section is *the representation of equivalence between senses.*

Two methods for doing this will be compared. In one, called the *Intermediate Concept Space Representation* (ICSR), there is held to be a language-independent set of concepts which are realized in differing languages each with the appropriate term. Figure 1 shows some equivalents between Schaufel (German), Aube (French) and Vane (English), which are appropriate in the field of Astronautics. It must be noted that what we have called "senses" above are represented by their term-names only both in the figures and in the text; this device is used merely for clarity of exposition.

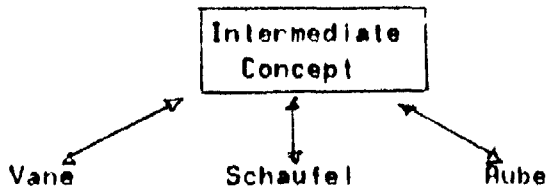


Figure 1: ICS Representation of
Vane=Schaufel, Schaufel=Aube & Aube=Vane

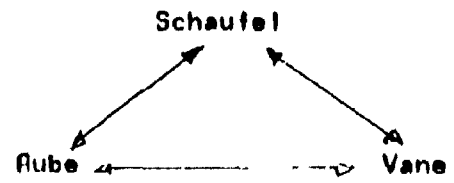


Figure 2: DE Representation of
Vane=Schaufel, Schaufel=Aube & Aube=Vane

In an alternative method, the *Direct Equivalent Representation* (DER), there is no need for such an intermediate concept space. Each sense accesses its equivalents directly, as shown in Figure 2.

ICSR is attractive because it offers a conceptual elegance absent from DER -- there is a universe of objects, each of which has a different linguistic representation in each language. This is a hypothesis about the nature of language which is known to be a misleading oversimplification for everyday usage, but its proponents presumably hope that it could turn out to be sufficiently true for the more restricted domains of specialized terminology

The two major ways of comparing these two alternatives are (1) in terms of the computer space taken in holding them and time taken in retrieving them, and (2) in terms of their adequacy when the dictionary must be modified The former indicates that under some circumstances ICSR can be cheaper in terms of space, but the latter shows that DER is resoundingly more adequate for the task of representing equivalence, and thus was chosen for Target.

4.1 Space and Time Analysis

Each of the lines in Figures 1 and 2, whether between an Intermediate Concept and a sense (Figure 1) or between two senses (Figure 2), represents what is called in computer parlance a *pointer*. Pointers need space in the computer and -- perhaps more importantly -- take processing time when used. Thus to get from "*vane*" in English to its German equivalent (*Schaufel*) requires the use of two pointers in Figure 1 (ICSR), but only one in Figure 2 (DER) This greater efficiency of DER is true for all pairs of equivalents.

Differences between DER and ICSR so far as space is concerned depend upon the number of languages in the multilingual dictionary. If there are *N* languages attached to an Intermediate Concept, there will be *N* pointers, one to each In the worst case for DER, each of the *N* will have a pointer to all of the *(N-1)* others, requiring *(N*(*N-1*))/*2* pointers Since there are three languages in Figures 1 and 2 (English, French and German), *N* is 3 and hence there is no advantage for either DER or ICSR The larger *N* becomes above 3, the greater is the advantage for ICSR; for instance, if *N* is 5; then in the worst case DER requires twice as many pointers as ICSR and if *N* is 7, DER requires 3 times as many in the worst case This worst case occurs when equivalents are present in the dictionary for *every* language, which may not be so in practice, especially while a dictionary is being compiled In the most favorable case, *N=2* and

DER has the advantage by a factor of 2. Dictionaries prepared for American use may often be English-X, so that $N=2$ and DER has a space advantage as well as the time advantage demonstrated above.

4.2 Modifiability

Irrespective of these considerations, a dictionary must remain functional while it is incomplete. To be realistic, it is probably uncommon for a dictionary to be "finished", and all automated dictionaries must be built incrementally, equivalent by equivalent. There are important differences between ICSR and DER, both in processing when equivalences are entered and when using an incomplete dictionary.

We need only consider as simple a case as the structures of Figures 1 and 2. Let us suppose that none of the equivalences *vane=Schaufel*, *vane=aube* and *Schaufel=aube* have yet been inserted in the data-base and they must be inserted. After entering the equivalence *vane=Schaufel*, ICSR will look like Figure 3a and DER like Figure 3b:

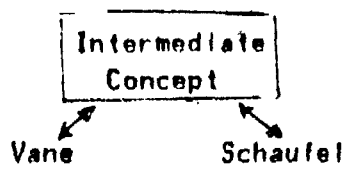


Figure 3a: Vane=Schaufel (ICSR)

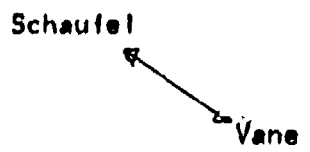


Figure 3b: Vane=Schaufel (DER)

(Note that this is a case where $N=2$ in the space analysis above, and so -- at this point -- ICSR has two pointers and DER only one.)

Now the equivalence *vane=aube* is to be inserted. With ICSR, the terminologist has no choice but to determine whether *aube* is equivalent to *Schaufel*. If they are, then Figure 1 is obtained. But suppose that they are not; then Figure 4a would be obtained:

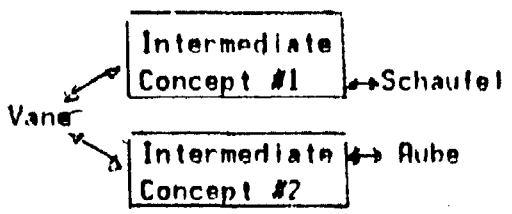


Figure 4a: Vane=Schaufel, Vane=Aube,
but Schaufel≠Vane

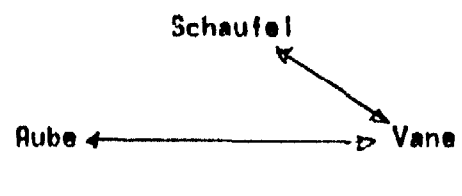


Figure 4b: Vane=Schaufel, Vane=Aube,
but Schaufel≠Vane

Note that for ICSR, the terminologist is forced to check *every existing equivalent* of a term when adding another, a procedure whose complexity increases exponentially with the number of languages. The competence of the terminologist must extend to *all* the languages in the database. With DER, on the contrary, no more need be done than simply adding the new equivalence as in Figure 4b. *Only if there is a known equivalence* between *Schaufel* and *aube* will the situation shown in Figure 2 be obtained.

A tempting, but incorrect, solution to this problem for ICSR is to *assume* that *aube=Schaufel*, producing Figure 2 whether or not it is actually appropriate. This is a

kind of risky and uncontrolled inference which ICSR can naturally force upon the user. There may be subtle differences in meaning between languages, yet ICSR forces transitivity of the relation of equivalence between all langauges. There is an alternative approach within ICSR, in which this is not the assumption, but this will lead to precisely the proliferation of senses which ICSR was designed to avoid. Furthermore, the simplification of intermediate concepts which are found to be redundant will be a complicated procedure.

In summary, the point so far is that the addition of an equivalence is a drastically more complex procedure in ICSR than in DER, and secondly that ICSR requires the terminologist to be as multilingual as the database, while DER does not. A further point may be made which concerns *inferencing*.

"Inferencing" means finding a near equivalent when an actual equivalent is not immediately obtainable. Of course, it is to be hoped that an automated dictionary will usually have an *immediate* answer to a user-request for an equivalent, in the sense that the requested equivalent has previously been entered explicitly. However, situations will occur where an immediate answer is not available. In that case, some form of *inferencing* may help. With ICSR, that inferencing has already been done in setting up the intermediate sense by means of the assumption above, and thus the information that it is an inference is lost at retrieval time. With DER, the pointers must be followed through explicitly and thus the system can report to the user the extent of the tentativeness of the derived near equivalent.

The disadvantage for the Intermediate Concept Space Representation, then, is that on the one hand finding an equivalent always takes *two* pointers, while Direct Equivalent Representation needs only *one*, and on the other -- more importantly -- DER is more able to represent nuances of meaning across languages and incomplete states of the dictionary. Hence Target uses the Direct Equivalence Representation for term equivalence.

fraise a fileter thread mill (EN)
Iron Steel Industries: Machines and Apparatus;
nop

fraise conique countersink (EN)
Iron Steel Industries. Machines and Apparatus;
nop

fraise-mere hob (EN)
Iron Steel Industries· Machines and Apparatus;
nou fem

fraises milling cutters (EN)
Mechanical Engineering Machines for Moving and Processing Materials;
Iron Steel Industries: Machines and Apparatus;
nou fem plu

frais fixes fixed costs (EN)
Economics;
Reference Terms. frais
nop plu
Usage Sample ...une nouvelle augmentation des frais fixes... [Kre4376]

frais generaux overhead charges (EN)
Technology and Industry In General;
Financial Affairs - Taxation - Customs;
nop
Usage Sample . . . les frais generaux (frais administratifs, de personnel et de gestion des polices d assurance). [SGB6/77]

frittage fritting (EN)
Iron Steel Industries· Pig Iron Production;
Mining Preparation and Refining of Raw Materials From Mines;
nou mas
Definition: roasting process in steelmaking [fr78]

frottement friction (EN)
Iron Steel Industries: Stress-relieving Deformation;
General Terminology;
nou mas