# An Examination and Confirmation of a Macro Theory of Conversations through A Realization of the Protologic Lp by Microscopic Simulation

A thesis submitted for the degree of Doctor of Philosophy

by

Paul A Pangaro

Department of Cybernetics, Brunel University

May 1987

## ABSTRACT

Brunel University, Uxbridge Department of Cybernetics

Paul A Pangaro

An Examination and Confirmation of a Macro Theory of Conversations through
A Realization of the Protologic Lp by Microscopic Simulation

1987

Conversation Theory is a theory of interaction. From interaction (the theory asserts) arises all individuals and all concepts. Interaction, if it is to allow for evolution, must perforce contain conflict, and, if concepts and individuals are to endure, resolution of conflict.

Conversation Theory as developed by Pask led to the protologic called Lp which describes the interaction of conceptual entities. Lp contains injunctions as to how entities can and may interact, including how they may conflict and how their conflict may be resolved. Unlike existing software implementations based on Conversation Theory, Lp in its pure form is a logic of process as well as coherence and distinction.

The hypothesis is that a low-level simulation of Lp, that of an internal and microscopic level in which topics are influenced by "forces" that are exerted by the topology of the conceptual space, would, in its activation as a dynamic process of appropriate dimension, produce as a result (and hence be a confirmation of) the macroscopically-observed behavior of the system manifest as conflict and resolution of conflict. Without this confirmation, the relationships between Conversation Theory and Lp remain only proposed; with it, their mutual consistencies, and validity as a model of cognition, are affirmed.

The background of Conversation Theory and Lp necessary to support the thesis is presented, along with a comparison of other software approaches to related problems. A description of THOUGHTSTICKER, a current embodiment of Lp at the macro level, provides a detailed sense of the Lp operations. Then a computer program (developed to provide a proof by demonstration of the thesis) is described, in which a microscopic simulation of Lp processes confirms the macroscopic behavior predicted by Conversation Theory. Conversation Theory thereby gains support for its use as a valid observer's language for every-day experience, owing to this confirmation and its protologic as a basis for psychological phenomena in the interaction of conceptual entities of mind.

# Table of Contents

# Acknowledgments

There are many individuals who must be thanked for their help in the research and production of this dissertation. First and foremost is my thesis advisor, mentor and friend Dr Gordon Pask whose intellectual and spiritual life have been the greatest influence in my career.

To the many individuals who made up System Research Ltd over its long existence my thanks to them must be anonymous. My appreciation is especially strong for those who suffered the pressures of its research programme and research conditions and who may or may not be individually identified for their very tangible contributions to Conversation Theory. Elizabeth Pask provided emotional support and personal expression of a kind that is rare in the world and without which I could not have persisted at System Research.

Mr Colin Sheppard of the UK Admiralty Research Establishment (ARE) provided contract support for the construction of THOUGHTSTICKER at a time when its subtlety and power could be seen only as a concept. He must be acknowledged and thanked for continuing the type of crucial and discriminating support championed by Dr Joseph Zeidner, who during his tenure as Technical Director of the US Army Research Institute supported the work of Pask for its own sake. Mr Dik Gregory also of ARE provided intellectual support and contributions to the construction of THOUGHTSTICKER during its development.

Dr Jeffrey Nicoll while Director of Research at PANGARO Incorporated constructed the complex innards of THOUGHTSTICKER and hence conquered both the Symbolics environment and the work of the Pask on the subject of Conversation Theory. He has also contributed to its formal and theoretical side. He provided undaunted moral support for my efforts

on the dissertation and continues to be an important collaborator and close associate.

Mr Peter Paine as my dual in PANGARO Limited has provided strong support and has allowed me to take the time and resources to complete this dissertation, often to his own disadvantage when timescales and responsibilities of contracts were very great.

Others who provided moral support without which I could not have completed are Herbert Brun, Patricia Clough, Graham Copeland, Karen Rose Elder, Michael Granat and Symbolics Education Services, Christina Gibbs, Heather Harney, Kevin Kreitman, Shelby Miller, Abe Raher, Vivian Scott, Louis Slesin, Ricardo Uribe, Eric Wolf, and especially Heinz von Foerster, for his foundations for Conversation Theory and the untiring vitality he has expressed to me.

# 1. Preface

## 1.1 Structure of the Dissertation

In the writing of this dissertation, while covering the necessary points on the main issues of the thesis itself, I was encouraged by my colleagues to insure that I had provided the following elements:

1. Background on Conversation Theory itself.

2. A personal history of my involvement with Conversation Theory, including why I had adopted it as an approach to the problems that interested me.

3. An indication of my own contributions to the field.

Upon review of the drafts up to a certain point, I thought that Point 3 had not been adequately expressed. I believe this had been the case in part to keep a dispassionate tone in a scientific work. Also, I specifically did not want to overstep a basic humility by giving attribution to myself as a single individual where the ideas were so much the combination of past efforts and more recent expressions of individuals other than myself. It is particularly awkward to make such differentiations in the context of a cognitive theory which emphasizes the ever-shifting definition of "individual" based on beliefs rather than biological identification. The theory also discourages attribution by providing a detailed model of how new ideas can arise only from the seeds provided by others, in a dimensionality of time that is neither linear nor fully ordered. However I can state that the core of the thesis is entirely my own, namely, that the addition of the process component to software manifestations of Conversation Theory provides a confirmation of important, predicted and

otherwise unconfirmed cognitive features. The software written to prove this by demonstration is solely mine.

Point 2, concerning a personal history of my relationship to the Theory and the context in which I adopted it, is fulfilled from a personal perspective in the next chapter, and from a software development perspective in Appendix C. It has been rewarding to reconstruct the personal side and to express, albeit *post hoc*, how my career has proceeded from the ideas rather than *vice versa*.

Point 1, concerning background, is less direct because the story to be told cannot emerge as a simple narrative. The subtlety and scope of the meanings require a hermeneutic circle. This cycle of interpretation is expressed in the body of the dissertation as starting with my personal interest in the Theory and related techniques (Chapter 2), moves to the Foundations of the thesis in Conversation Theory (Chapter 4), and uses the software of the Theory to explain its elements and procedures in detail (Chapter 5). Then (Chapter 6) the limitations of past software is revealed and the true operations emerge by the addition of the process component of the Theory. The summary (Chapter 7) is a recapitulation of the main points and possible extensions. This is followed by Appendices, with the technical details of the implementations, Bibliography, Glossary and Figures. As an Annex to this dissertation the THOUGHTSTICKER User Manual (Pangaro *et al* 1985 in the Bibliography) is attached for completeness.

The explication of Conversation Theory within the text is thus achieved only upon completion of the cycle whereby the symmetries and aesthetics noted in the beginning are achieved by an innovative approach to implementation which fully explores the central tenets of the Theory.

I believe that all of the requirements are therefore fulfilled and I hope the

result is an effective examination of both the original Theory, and its confirmation and extension represented by this dissertation.

## 1.2 A Context for AI and Cybernetics Terminologies

With the surge of interest in the field of Artificial Intelligence (AI) primarily due to technological advances since 1980, certain concepts have gained acceptance and comprehension within a wide audience of researchers and software development projects in academia, industry and government. Because of this, terms such as "knowledge elicitation", "knowledge representation" and "machine reasoning" now have common meanings and provide a background in which discussions in those communities may take place (Barr & Feigenbaum 1981).

Each of these ideas had been given full treatment with detailed meaning and context for interpretation within Conversation Theory (CT) as developed by Pask and others (Pask 1976a, Pask 1980a), well in advance of their recent uses within AI. Unfortunately, up to the 1980s, CT received wide exposure only within the fields of cybernetics and computer-aided instruction. Within those spheres and as illuminated by CT, many core concepts of epistemology and human discourse were given tangible meanings that both reflect a common sense usage and a precise and (within a cybernetic interpretation of the term) scientific meaning. The terms "individual", "conversation", "agreement" and "understanding" are prime examples of this (Pask 1975a).

AI, engaging many more researchers and hence research publications, is perforce divided in opinion and much more fragmented in technique than CT. (It is an editorial comment to note that the fragmentation is evidence of the lack of coherence and direction in the field.) Thus a perverse situation has arisen where consistent and agreed meanings within

Conversation Theory cannot be explained using terms from AI without both distortion and ambiguity. And, common every-day terms cannot be used unqualified to describe Conversation Theory without losing a freshly-new and yet scientifically-strict meaning. Therefore, although I will often draw on the metaphors of Artificial Intelligence I will endeavor never to do so without immediately providing the significant difference to the realm of Conversation Theory.

In general and to avoid constant qualification, references to AI do not indicate the entire field of Artificial Intelligence, but rather those areas within AI that relate to the subject of this thesis, namely, knowledge representation and machine intelligence. None but the most obstreperous proponent of AI will object to this usage.

# 2. Background

## 2.1 A Context for Adopting Conversation Theory

In 1976 I was engaged in software research projects centering around the use of highly interactive computer graphics systems which in the present day are taken for granted on any home video game; in those days such equipment was extremely rare as it was only just being developed. The Architecture Machine Group, a research facility at the Massachusetts Institute of Technology, was producing innovative hardware systems for the creation of new types of media environments: large screen displays, many simultaneous auxiliary displays, touch panels and tablets for input of commands, graphics and even gestures. The work of this laboratory has influenced a generation of workers in the field of interactive computing. Its name must not be confused to imply so narrow a field as mechanical architecture; rather, it was concerned with the influence of mechanical and electronic and digital artifacts on all aspects of the "built environment."

### 2.1.1 The Needs of Man-Machine Interaction

My background and interests at that time were centered on the issues of man-machine interface (MMI) specifically for the creation of computer graphics. These visual results might be static or dynamic, but always for the purpose of expressing ideas, whether to oneself (as an aid to the process of design) or to others for the purpose of communication. At MIT I had already had the privilege of access to the newest and most powerful computer graphics systems anywhere; what I felt was lacking was a powerful framework in which to express the problems of MMI.

It was my conviction that to make a machine produce images representative of abstract ideas:

- There should be a close connection between the formulation that the user conceives on the one hand, whether in diagrams, pictures, movement, *etc.*, and the gestures made to the machine, whether in typing text, programming, drawing, whatever, on the other.
- All of the power of "programming" should be available to the designer/user, in the sense that procedures and conditional branching could be used to great advantage, for general modelling as well as the conveniences of repetition and variation.

The combination of these two ideas, and exposure Pask's protologic, led to my design for a visual programming language for simulation-based graphics of great expressive power (implemented by a research team and described in Pangaro, McCann, Davis & Steinberg 1977, and Pangaro 1980).

One common paradigm of the era was that the human's task was to tell the machine what was required. I however felt that this was not a complete image; that it was also the requirement of the machine to tell the human what could be done. These requirements were not fixed or ordered in time. They would vary depending on the background and needs of the human as well as the machine. The system's capabilities evolved (although not in the same time frame) in parallel to the human's, as new versions of system code or new capabilities were made available. Hence requirements would emerge over time, rather than be done "all at

once" at the start of the interaction. It seemed essential to me that insofar as needs and knowledge evolve so must the interaction.

Therefore it was clear to me that a kind of teaching/learning communication was necessary, and one which was symmetric: both the human user and the mechanical machine had to both teach and learn.

## 2.2 Available Models before Conversation Theory

Obviously the interaction between human and machine was much more limited than that between human and human; but I imagined that since one limiting requirement (in some important aspects, perhaps the main one) was that of the human and hence the human to human model might be a useful place to start. Surely there was enormous history, cultural and scientific, technical and artistic, on that subject.

### 2.2.1 Shannon's "Information Theory"

From the scientific community, Shannon's communication theory (Shannon & Weaver 1964) seemed to be the only direct foray into this problem, especially in that it was named to address this very problem. The conception here is that communication involves a channel between entities playing roles (perhaps alternately) as "sender" and "receiver." The concern of the approach is to control the uncertainty with which a "message" is transmitted across the channel. Transmission is defined as the correct receipt of a sequence of encoded data which makes up the message. Variation in the noise of the channel determines a statistical measure of "goodness" of the channel. Much can be said by communication theory about the redundancy required to insure a given and desired level of certainty about the datums [*sic*] getting through unaltered.

Given the robustness of the formulation and the major concern for insuring the accurate (indeed "perfect") data required for computers to operate (especially in the era of the 1950s when the limits of performance of vacuum tubes were being reached) this approach was a landmark for many of the problems in communications and computers.

Application of this model to human conversation however is fraught with compromise and difficulty:

- The data are exactly that, *data*: objective encodings or symbols that stand alone, require no context, and are either one symbol or another, unambiguously.

- The class or alphabet of symbols is a fixed set and cannot be expanded; the ability to recognize one from another is predicated on the need for both the sender and receiver to have agreed on the fixed set beforehand.

- The redundancy described exists within the encoding scheme as applied to symbols; there is no bearing on the redundancy realized by the interpretation of the message as a whole.

These objections individually and together remove the utility of the approach for application to human discourse. This work remains a foundation in branches of "communication technology", true, but at the practical level it serves little more than to express some technical issues associated with bit transfer in hardware channels. Weaver (in Shannon & Weaver 1964) admits to multiple layers of interpretation to the problem of "communication theory":

## 2.2.2 Artificial Intelligence

Much had been said by this time about "intelligent machines"; the field of AI had already been through a number of cycles from promise to difficulty to redefinition of promise (Feigenbaum and Feldman, 1963; Minsky 1968; Nilsson 1980). Despite the difficulties the focus of the field remained (and does to this day in the latter 1980s) on the "intelligence in the machine" [*sic*]; little or nothing is said about communication with such an intelligent machine or between man and machine. And this was for me the precise focus of need as I formulated it then: solutions to the problems of communication must be part of any solution for machines of intelligence.

AI has always seemed to me based on an over-confidence in Turing computability (Minsky 1967). This has been supported in arguments by Jerry Lettvin in which he specifically ties the AI community to work by McCulloch and Pitts on the equivalence of simplified threshold networks to Turing computability (Lettvin 1985).

The coupling of these two mathematical results unfortunately allowed the AI community to avoid questioning its foundations, based in the presumption that the power of Turing mathematics is supreme (a mistake Turing himself did not make, as I learned by examining his unpublished works at Kings College Cambridge). This over-confidence has prevailed until recently when AI, physics and cybernetics were united in new work to extend the definition of computability (Deutsch 1985). These extensions were presaged by Pask (*e.g.* Pask 1958).

Born and raised on a mathematical formalism and whatever technological capabilities that followed, AI could not see that it could not see its limitations (to paraphrase von Foerster). Cybernetics was simultaneously proceeding from an epistemological basis of what can be known and,

especially in CT, moving to theoretically sound and practical formalisms on the nature of knowing. More detail on how CT accomplishes this is given later in this chapter.

## 2.3 Reasons for Adopting Conversation Theory

I was introduced to Conversation Theory first in the form of Pask himself, who was consulting for the Architecture Machine Group. Pask had influence there by critiquing research programmes, inventing metaphors and providing a rich interconnection with other workers in many fields, which he brought to a Group concerned with increasing the bandwidth (my term) of interaction between human and machine-based systems. One tangible result was collaboration of the entire group (and fortunately myself included) in the production of a major work called Graphical Conversation Theory (Negroponte 1977). This was a research proposal submitted to the US National Science Foundation, which would interpret CT in light of the newest and most powerful computer technology. (Alas it was not funded.)

These interactions led me to the study of Pask's papers, frequent visits to his research laboratory, and eventually to collaboration on research projects. It was this collaboration under contract to US and UK establishments that funded the implementation of THOUGHTSTICKER described in this thesis.

It is a subtle task to separate out a set of personal, individual reasons for my becoming interested in CT, or for using it as the basis for endeavors in computing, or for using it as the foundation of a research dissertation in cybernetics. With the understanding that any such delineation is for descriptive purposes only, here follows an attempt to linearize what must be, as its origins in CT would tell, a set of reasons that are ultimately holistic and hermeneutic.

### 2.3.1 Symmetry across Individuals

CT restores a symmetry to the modelling of all interaction. No hierarchy exists between, for example, teacher and learner; both must "teach" and "learn" from the other in order to achieve communication or, as is preferred within CT, **conversation** (see the Glossary for a definition). These interactions are considered to be "I/you" referenced, because one individual treats the other as of equal rank, in that the language is one of command and question; the other individual has options and may or may not respond, cooperate, *etc*. This provides an aesthetic as well as an ethical formalism (Pask 1980b).

### 2.3.2 Symmetry within Individuals

CT models discourse within individuals by levels which are symmetric to each other and to those in other individuals. The model stratifies any language of discourse into distinct levels (at least from the perspective of the observer) and creates dependencies between these levels. Thus, a "higher" level **determines** the actions at the level "lower" to it. These interactions are considered to be "it" referenced because no choice or response is allowed by the "lower" level (Pask 1975b).

One consequence is that given a level considered to be one of "method", the lower level is where that method is carried out. Thus, the lower level is an "environment" for the "higher" level. Consider that the environment may be an external world of actions, or merely **further levels of cognitive activity.** This symmetry provides aesthetic satisfaction as well as an implication that computation can encompass actions in a world of physical objects as well as mental constructs. This interpretation served as the basis for my design of the Expertise Tutor, a software prototype developed under contract to the UK Admiralty which contains precisely

this multiple level of discourse and access for the user. It is the first system of which I am aware which makes this distinction of levels both explicit and accessible to the expert and user alike (see further description in Appendix Section C.7).

### 2.3.3 Subjectivity and Objectivity in the Same Framework

The above two points provide a brief description of the "conversational framework", a structure in which scientific observation can be made and descriptions of interaction may be given. The framework provides for both "objective" interaction, where no interpretation is made (relative, as always, to an observer) and "subjective" interaction, where any result can be seen only from a context **within** the interaction of the two (or more) individuals.

Scientific discourse has always insisted in "objective" enquiry. It is this very insistence which has kept psychology out of the realm of mental activity (by its own admission). CT provides, I think uniquely, a framework in which objective, "hard-valued" measurement can be performed in the domain of mental activity. For example, a hard-valued, objective and scientific meaning for "agreement over an understanding" is obtainable within CT (Pask 1975c). Because of this, the requirements of MMI for the transfer of information about a system's capabilities and a user's desires can be specified.

### 2.3.4 The Language of Conversation

The interactions described must occur in a language, and here is the crux of any framework. If the language is a "natural" one, such as English, immediately any machine interaction is disqualified. Although it may

appear that our present-day interaction with computers is "in English", in fact English words and phrases are used merely as tokens to indicate constant and pre-determined meanings. No interpretation is involved and hence the use is not of "natural" language.

It may therefore seem that, similarly, CT is inadequate for any advances in MMI. This is not the case for two significant reasons:

1. CT as a framework is adequate for any language so long as it is one of question and command (von Wright 1963). It may be gesture or dance, visual or aural, images or imagery.

2. The use of language tokens can be kept at a mechanical level within the software, with the user providing a "semantic" value to it. Interpretation is brought in when a user relates topics together to formulate the "meaning" of the relationship (as in the CT construct of a "coherence", detailed in Section 5). The activity is basically hermeneutic and the meaning arises in the circular interpretation and use of tokens by the user.

### 2.3.5 Generality of "Individuals"

Interactions occur across an interface, among individuals. The distinction among individuals is made by the observer, who asserts the existence of the interface. The individuals so distinguished and the observer can be considered as duals of each [*sic*] other. The emergence of a distinction among individuals comes at the moment of distinction between self and other, who are the same type of individual.

In CT, individuals are "P-Individuals" or psychological individuals, rather than a simple reduction of physical individuals. Thus a single human may be modeled as consisting of many P-Individuals, different at different times, for varying purposes, evolving in the course of experience. This would encompass the requirement for a design of user interface software which is adaptive to the changing needs of the user, in a variety of guises and situations.

Similarly, the same framework can be used to model an interaction between a human and machine. The specifics of processes that are available within each are clearly different; however they can be specified and the resulting needs for mutually-understood interaction are achieved.

### 2.3.6 Cognitive Bases of CT

CT was developed not out of whole cloth but from a history of empirical research on the nature of interaction, conversation and understanding (Pask 1975a, Pask 1975c). The theory which resulted therefore incorporates its origins into its terminology and structures. The terminology has a great deal of "common sense" appeal and the theory provides explanatory support for many everyday events (including forgetting, remembering, mnemonics, confusion, ambiguity, uncertainty, and conflicting desires). This is particularly evident when contrasted with competing theories (*cf.* Minsky 1986).

### 2.3.7 Mediation of Language by a "Knowledgebase"

Knowledgebase is a term which is used with abandon in the field of AI to mean a structure internal to a computer which "contains knowledge" and which can be manipulated, perhaps to make inferences or deductions, in software. CT maintains primary interest in a "knower", while the "knowledge" cannot be held as independent from such an individual.

CT defines related structures in its dual called Lp (pronounced "L-sub-P" and explained in detail in the course of the text). Completely consistent with CT and all of the points described above and below in this text, Lp is a class of well-specified processes that operate on a class of well-specified structures that can be adequately computed in present-day, serial digital computers. ("Adequately" is a point taken up later and the distinction between various levels of simulation of Lp is a central point of the entire thesis.)

## 2.4 The Emergence of this Thesis

Given of the above, it seemed extraordinary that there should exist a theory of aesthetic elegance, simple formal symmetries, based in cognitive behavior, **and with a detailed calculus of knowing that could be programmed.**

Preceding sections describe the conditions under which I was introduced to CT, especially in the context of MMI. My interests had always been considerable, however, in the nature of cognition and communication, and how computers may enhance or otherwise influence these daily human activities. I have often heard others working within CT and cybernetics say that they had some affinity or intuition for the ethos beforehand; upon introduction, the expressiveness of the framework was immediately apparent. The simple elegance of CT to describe mental events, and its coherence with the arts and humanities (Pask 1968, Pask 1976b) as well as sciences (Pask 1979), were a constant source of interest for me. My initial entry from the perspective of MMI grew into a general interest in its tenets. The desire to influence an entire field with the sweeping power of CT by producing computer-based implementations of its ideas became (and remains) very strong.

All of the above reasons drew me into the world of Conversation Theory and each successive revelation within it confirmed to me its power and utility for my interests, both theoretical and practical.

Any such software based on CT, to be useful in commercial applications (namely, every-day use) and to be of sufficient power to influence the world's view of MMI according to the ethics of cybernetics and CT, would require considerable investment and relatively single-minded course of activity.

Since my first exposure to CT, I have devoted considerable time to designing and coding software systems based on its tenets; details of my own and others' contributions can be found in Appendix C on the history of THOUGHTSTICKER. (That it also required the creation of a company framework is a detail of management and of politics.) It was in the course of development of this software that two issues converged: the need for maintaining the process component in the simulation of the calculus of Lp; and the evolving display of the Lp structures for the sake of the user. This discussion is taken up in detail in Chapter 6.

This thesis returns emphasis to the importance of the process component of Lp in any research and development centered on CT. In a very real sense, without process the theory is lost, as one of its trilogy of features is missing (the others being distinction and coherence). This is due to the central role that process plays; for example, that CT states that memory is not recall of a static configuration but a dynamic recalculation or reproduction. All of the formal expressions of relationships with CT contain production arrows which are not merely transformations from state to state, but continuous processes whose continued execution and persistence is the given cognitive element (topic, memory, concept). Hence the process component is key, and one contribution of this thesis is the reinstatement of that component to software manifestations of CT.

# 3. Review

## 3.1 Knowledge-Based AI Systems

Once I made the shift from principles of MMI to general theories of cognition, it was necessary to ask the question as to whether, in all of the techniques developed in the field called AI, some of its ideas or software results might be appropriate, useful, or better than those of CT.

### 3.1.1 Semantic Nets

Semantic nets (Quillian 1968) appear on first review to be closely related to Lp structures; an early question often posed in discussions about CT with AI researchers is, How are Lp structures different? Because of this apparent (but not actual) close association, a brief description follows. (Details of Lp structures can be found in Chapters 5 and 6.)

Semantic nets consist of nodes and links. Nodes refer to objects or attributes, linked by arrows (or, in programming terms, pointers) which have values in themselves. For example, [FRED IS-A BIRD] relates the nodes FRED and BIRD by the link IS-A. There must be many types of links, covering ideas such as ELEMENT-OF, HAS-PART, GENERALIZATION-OF, EXAMPLE-OF, and so forth. If in doubt, you simply create a new link willy-nilly.

The ability to create new links seems to provide for a general scheme without boundary. However this very generality is its downfall. The class of link types becomes very large and it rapidly becomes apparent that any subtlety or power of the scheme is simply shifted one level into the operators that the links represent. The nature of the computation contained in the links (such as generalization, inference, and so forth) is not well-specified.

It is beyond the scope this text to explore this question with too many specifics; however, there are clear differences which can be briefly listed and which provide justification for choosing Lp above semantic nets in the my research efforts that followed the initial enquiries:

1. Semantic Nets emerge primarily from programming constructs; they have some common sense appeal but no basis in cognition or empirical research.

2. There is no theory of knowing which shows that semantic nets are minimal, necessary, sufficient, or even useful representations of human knowing.

3. Further refinements (Brachman 1979) to the approach have added considerable complexity but without achieving major advances or overcoming the objections put forth even from its proponents (Brachman 1985).

Lp, as shown in this thesis, has none of these disadvantages, and considerable advantages.

### 3.1.2 Frames

One major development (in a sense "on top of" semantic nets) is that of Minsky's Frames (Minsky 1975). This approach accumulates semantic relations (in the sense as shown just above) into frames of knowledge that are related to contexts of interpretation; for example, while in a restaurant, while going to a play, *etc*. This refinement handles cases of "default" knowledge, where the scheme can attempt to fill in about items not explicitly explained (a simple form of generalization). Unfortunately

the old problems remain; each of the above objections to semantic nets could be paraphrased to apply to frames. Even more one is given the feeling that these are structures that are conveniently computed by programming languages such as LISP, and hence their popularity within AI.

Minsky has most recently revived his concept of the "Society of Minds" theory of cognition (Minsky 1986). (This idea and others contained in a paper called "Consciousness" were circulated privately in the MIT AI community in the 1970s.) Basically, the society of minds puts forth the idea that a mental organization consists of many, possibly conflicting sub-units. These smaller units each require resources to be computed and provide competition for the limited resources. The approach is intended to address (what I will call) "post-Freudian" problems. These are Freudian, because they deal at the psychological level identified by his followers as the concern of Freud. They are also "post-" because they are the interpretation of Freud rather than Freud himself.

Minsky offers engaging argument but neither theory nor confirmation of his ideas. In fact, considered as metaphor and ignoring the Freudian overtones, Society of Minds is quite consistent with CT's modelling of the "P-Individual" being the unit of perspective within a mental organization, conversing (and competing and conflicting) with other P-Individuals in the same organization. Pask however provides details of:

- How the P-Individual is composed, *namely, processes that* can be modeled by Eigen functions.
- The means by which they converse, namely a language capable of question and command.
- How the interaction can be modeled, namely the "conversational paradigm" (Pask 1975b).

- A detailed model of the structures that make up the transactions, *i.e.* the Lp calculus.
- How conflict and its resolution can be modeled, *via* Lp and the operations described in the main body of this thesis.
- How a continuing process of "saturation" occurs, forcing the interaction of otherwise independent cognitive structures which in turn creates new structures or reveals conflict, ambiguity, confusion.

Given the specificity of CT and the delightful but vague and unfulfilled images of Society of Minds, the decision to use CT to attack problems of my interest was a simple one. In fact, it was just such a formulation that caused me not to pursue my work in a doctoral programme at MIT to which I had been accepted, in order to pursue the line of research described in this dissertation.

### 3.1.3 Expert Systems and Rules

Expert systems have received major attention most recently. These utilize "production rules" in the form of "If...Then..." statements. For example, "If the temperature is above 50 Celsius and the smoke detector has been set off, conclude there is a fire in the room." Such statements are said to represent the knowledge of experts, and to provide the means to model how experts actually make decisions. Statements are processed together to create new conditions that "fire" other rules, which fire yet further rules, *etc*: "If there is a fire in the room set off the fire alarm and the sprinklers." Given that some rules represent desired conclusions that are distinguished from others, the system is said to "decide." Alternatively,

the expert system can work backward from conclusions to necessary pre-conditions and thereby diagnose initial causes.

PROLOG is a programming language designed to process these descriptions of "knowledge", and the general approach has its origins in first-order predicate logic. Comments about semantic nets still apply: the approach is not based on cognitive theory or empirical studies of human knowledge; the scheme is not known either to be sufficient or necessary to explain human cognition; and extensions do not solve fundamental problems with the approach. Expert systems have recently become a popular means to approach the problems of training, wherein tutorial strategies are encoded as "If...Then..." rules: "If the student has failed test A and test B then conclude topic X not understood." At some point there should be a general recognition of this fashion as no better than an intricate but equally ineffective form of training as programmed instruction (in the same way that programmed instruction is now widely recognized to be an impoverished technique of computer-aided instruction; see Section 3.3.)

Other AI approaches are more tangential to the requirements of a cognitive approach to software and MMI design. Work in natural language parsing is still focused largely on translation and getting knowledge "into" a knowledgebase (Barr & Feigenbaum 1981). Shank's work on Scripts (Shank & Abelson 1975) has some interest in communicating with users, but although he takes an increasingly iconoclastic view of other approaches within AI (Shank 1980) his alternatives are still within AI's limitations. Winograd is the closest to a cybernetic view but his publications do not provide a sufficiently tangible alternative to begin coding (Winograd & Flores 1986).

## 3.2 Related Work in Cybernetics

Despite its popular associations with robots, cybernetics does not of itself refer to computers. A surprisingly small amount of work in cybernetics overlaps with, or has produced approaches to, MMI or conversational software.

### 3.2.1 Foundations

At the theoretical level, related work in cybernetics has generally been a precursor to CT and/or provided a foundation upon which CT could provide the specific results that it does (for example, von Foerster 1960). The emergence of second-order (retold in von Foerster 1985) and reflexive interpretations of science (Bateson 1960) provides the beginnings within cybernetics of an approach to systems that is both scientific and subjective. However these foundations require interpretations, in both empirical studies and detailed formulations, before they can be translated into tangible prescriptions for action, which came only with Pask.

### 3.2.2 Laing

The utility of a reflexive view of interaction (again on the theme of the problems of MMI as discussed above) is most effectively presented in Laing 1966. The interpretation in the context of conventional MMI would be something like "I [the user] know what functions the system knows. The system knows nothing about me." A more advantageous approach which I desired would be something like "I [the user] know that the system knows what I know about the system." This could perhaps be extended to incorporate goals, as in "I [the user] know that the system knows my goal is to ..." Thus the user could proceed with greater

confidence and efficiency. Laing thus provides a metaphor of desire, but nothing detailed on which to base a software approach.

### 3.2.3 Personal Construct Theory

In terms of software, the work of Kelly in the extraction of grids of constructs for purposes of explicating knowledge otherwise internal to a knower (Kelly 1966, Bannister & Mair 1968) is closely related to the interests of CT. This has powerful implications as seen in practical and modern software implementations (Personal Construct Theory and the software Pegasus, in Shaw 1980; and MAUD software, Humphreys 1975).

In these latter two cases, the software is used as a means of extracting constructs internal to the knower, and in a form which is self-consistent. This exactly parallels one of the intentions behind Lp, where the names of topics and the relations that they are contained in are delineated and named by the user. The software, again as in Lp, is used to reflect back to the user on the implications of the constructs and their structures, as for example in cases of ambiguity and contradiction (Humphreys 1980).

These other approaches both preserve the subjective quality of the "extracted knowledge" and emphasize the self-consistency of the result. However, Lp additionally provides a framework that is based on the epistemology of observation, empirical confirmation of the utility of its references to individual learning style (independently confirmed by Marante & Laurillard 1981, and Bogner 1986), and an extended set of operations which encompass many more events that are recognized as cognitive (Pask 1983).

## 3.3 THOUGHTSTICKER and Computer-aided Instruction

Computer-aided instruction (CAI) has been widely available on computers since the advent of minis and micros, starting in the 1970s. Largely accepted as useful tools for training by computer, some criticisms have arisen over the years (see Kearsley 1977 for a view inside the field, and also Pask 1972). The following section presents a self-contained explication of how THOUGHTSTICKER can be applied to the problems of a user learning from computers, in direct comparison to existing software training approaches. THOUGHTSTICKER represents a complete revision of all existing techniques.

### 3.3.1 "Intelligent" Training

THOUGHTSTICKER is an intelligent software system for training and information management. The system is "intelligent" in the sense that it mediates between an expert knowledgebase and a user to provide some of the features of human conversation: a shared vocabulary, history and context of the dialogue. It is the most effective system of its kind available on any hardware.

THOUGHTSTICKER was developed as an enhancement to conventional computer-based training (CBT) and provides substantial improvements to CBT in:

- Ease of use, for both courseware creation and delivery of training
- Management of the courseware creation process
- Sensitivity to individual learning style

- Training efficiency and effectiveness, especially in complex tasks
- Flexibility to encompass job-aiding and advising, as well as training.

The software consists of two independent parts: the means for creating the knowledgebase (the Authoring Module); and for giving access to the knowledgebase (the Tutoring Module or Browser). Both are conceived and implemented as generic solutions that can be tailored to the specific requirements of the application, its users, the target hardware and interactive media (including videodisc, CD-ROM, graphics and sound). THOUGHTSTICKER is attached easily to existing application software and simulations for a complete training solution.

### 3.3.2 Background of the Term "THOUGHTSTICKER"

The term THOUGHTSTICKER refers to software based on a cybernetic approach to the problem of measuring understanding in human conversations. In the 1970s, THOUGHTSTICKER was developed at Pask's laboratory as an extension of Pask's studies of the 1950s and 1960s in human learning and individual conceptual style. These studies culminated in a comprehensive approach to educational technology (the CASTE system) that has been widely influential in educational theory and computer-aided instruction.

The term THOUGHTSTICKER was coined by Pask to mark the maturation of a general approach to knowledge representation whose elements reflected cognitive structures. The name itself emphasizes that in order to converse we must externalize our thoughts into a tangible form for ourselves and for others. Using a computer as the medium for

this conversation means that thoughts must temporarily take a static form in the computer, before becoming dynamic again as they are interpreted by a user. THOUGHTSTICKER models mental structures with a few simple but powerful constructs that:

- Capture the author's or expert's precise approach to the subject matter, but still
- Allow the user to learn the subject matter according to his or her conceptual style.

THOUGHTSTICKER software is the medium for the conversation, not a participant.

The power of THOUGHTSTICKER derives from:

- A theoretical basis in cybernetics and learning theory. The advantages of Conversation Theory as a model for learning have been supported by experiments in cognitive style. THOUGHTSTICKER is derived directly from these ideas.
- Evolutionary development in application to complex training problems, including those with training in the performance of a task. Extensions for job aiding and expert advising have also been demonstrated.

### 3.3.3 Existing Applications

Prototype knowledgebases have been constructed by me and my colleagues in PANGARO Incorporated, and in the subjects of AI and

cybernetics, naval strategy, introduction to computer usage, and word processing.

For the Behavioural Science Division of the UK Admiralty, THOUGHTSTICKER has been integrated into an Expertise Tutor, consisting of a naval simulation and expert knowledgebase (described in Appendix C). The Tutor provides tactical training as well as basic rules and operations of the game. This system is effective because it provides the user with equal access to descriptive knowledge (elements, relations, goals), prescriptive knowledge (methods, tactics), and the environment (the simulation itself).

For the US Army Research Institute, a videodisc interface controlled by THOUGHTSTICKER has been developed to demonstrate training of a vehicle identification task.

Most recently a prototype training course has been developed for Symbolics Education Services. (Symbolics, Inc. is the manufacturer of advanced software engineering and Artificial Intelligence workstations; the most advanced implementation of THOUGHTSTICKER runs on this hardware.) Derived from an introductory, paper-based workbook written by Education Services, this course presents the basic components of the Symbolics computer, concepts of symbolic processing, and how to use certain features of the machine such as the editor and command processor. The learner can immediately practice what is to be learned *via* the "hands-on" capability: in the course of learning about the editor (for example) the editor window is automatically displayed and commands may be tried step-by-step by the learner concurrently with their presentation in the training material.

### 3.3.4 The User Experience

THOUGHTSTICKER facilitates the user in any training and information management activities by:

- Allowing a mixed-initiative dialogue so that the user may either give the system control, or direct the conversation based on immediate needs (*e.g.* uncertainty or current goal).

- Producing distinctly different actions and responses for different individuals, based on the background, purposes, context and cognitive style of the user.

Thus the user is provided with more focused and efficient interaction than conventional computer-aided instruction and information management systems.

These results can be achieved because THOUGHTSTICKER "models the user" throughout the interaction, creating a history with each individual that is maintained even across sessions. Because this user model is the basis of all actions by THOUGHTSTICKER, the interaction has more of the qualities of human conversation: context, focus, and shared vocabulary.

### 3.3.5 Comparison to Computer-Aided Instruction

The following two pages contain a brief, "side-by-side" comparison of conventional computer-aided instruction techniques and THOUGHT-STICKER.

## Conventional Computer-Aided Instruction

1. Based on concepts of "programmed instruction" developed in the 1950s and substantially unchanged since then.

2. The subject matter is given a pre-ordained sequence in which it is to be learned; there is no other structure to the material.

3. All users are treated identically, and thereby are presumed to have the same cognitive learning style.

4. The author of the subject matter makes assumptions of prior knowledge of the user; very little variation of material is possible despite differing backgrounds in the user population.

5. Additional questioning by the user is limited or not allowed. Remedial material is offered to the user upon supposition of reasons for user's failure and usually from a static model based on averages or likelihood.

## THOUGHTSTICKER

1. Based on a cognitive theory of human conversation developed over the period of 1955 to the present, and affirmed in empirical studies.

2. Uses a robust knowledge representation scheme to provide a true knowledgebase; all conceptual dependencies are represented in a network structure with no fixed paths.

3. Sensitive to an individual's cognitive style, modifying responses accordingly.

4. Sensitive to individual variation in user's prior knowledge and can be tuned by a variety of user profiles (for example, naive computer users; experienced computer users but not of this particular type; users of another particular vendors' hardware).

5. User is free to ask questions and explore throughout the knowledgebase at any time. The user helps direct the remedial dialogue, which is derived from a combination of user's focus, the structure of the knowledgebase, and the history of the interaction.

The comments about computer-aided instruction can stand as generalizations across a number of commercial products because they characterize a field which is substantially homogeneous. Although specific features of training packages vary, the instructional model and the organization of the subject matter does not.

The driving force of the interaction is the user's interests and uncertainties. THOUGHTSTICKER has specific features that help the user discover these interests and uncertainties, and then explore or resolve them. By allowing such strong initiative on the part of the user, THOUGHTSTICKER provides an effective, efficient and supportive training experience.

### 3.3.6 THOUGHTSTICKER's Training "Knowledgebase"

THOUGHTSTICKER is constructed as a generalized information management system. Its internal database, called a knowledge representation or knowledgebase in modern parlance, supplies a flexible, "relational" format that is suitable for any subject matter.

To describe the format briefly (to be detailed later in Section 5): topics are defined and associated in relations by the author or expert. These objects together define a network or mesh of "knowledge" and thus determine the structure of the knowledgebase. There are no pre-defined types of relation; the author is free to create relations as desired. THOUGHTSTICKER contains training heuristics, many concerned with the user's purpose and conceptual style, for moving over this structure. The conditions which determine the action of these heuristics are:

- The User Profile: A preset stereotype of the background of

the trainee. The author pre-determines what classes of users are expected to interact with THOUGHTSTICKER. For example, these classes may represent a particular range: novices at a particular task, individuals with some exposure to comparable tasks, and experts. The User Profile can be styled by the author as a single default state, or chosen from a descriptive list by the user, or determined with highest accuracy and detail from a pre-test. Given such a User Profile for a particular user, the choices THOUGHT-STICKER makes are more directed to that individual's level. However, the Profile is only a starting basis and the two mechanisms described next provide further refinement of THOUGHTSTICKER's actions.

- The User History: A tracking of all actions and results since the user started, whether at the present session on the machine or in the user's history with THOUGHT-STICKER over time. The history consists of, among other details, a record of terms used by the user and the system, topics and explanations shown, and the current context of conversation. This shared history is used by THOUGHT-STICKER at each moment to choose an explanation or a new focus of attention. The result is more directed for the user and hence more efficient and satisfying. The disk requirement for storing this User History is modest.

- The User Model: A representation of the user's conceptual learning style. As in the User History, the User Model influences THOUGHTSTICKER's choices at each moment, but by applying criteria associated with the user's preferred modes of learning. For example, these may include a preference for examples before general descriptions; or preference for thoroughly completing current areas of learning before touching on new areas; or preference for graphics over text. The User Model can be configured by the author, the user, or by the results of a pre-test. It can even be modified on the fly, provided the user is imposed upon to give feedback on the effectiveness of explanations. In addition, the User Model may include the broader components of the user's purpose. Thus THOUGHT-STICKER can respond differently if the user wishes to learn the entire subject, or the performance of a specific task, or a single precise command name.

### 3.3.7 Aids to Authoring

It is widely reported that the major expense in using computer-aided instruction is the cost of "authoring" the material, that is, creating the subject matter that the learner is to see.

Conventional computer-aided instruction provides basic utilities for the creation of text and graphics to be assembled into frames for the user to

view. In addition, features for managing the user's records, keeping statistics across groups, *etc.*, are generally available.

Like conventional CBT, THOUGHTSTICKER can provide any "management" functions relevant to a particular site; for example, tracking a student population, creating output reports, or collecting feedback on the effectiveness of any aspects of the course. These requirements are best defined for the specific needs of a CBT application, and tailored accordingly.

Unlike CBT, THOUGHTSTICKER is exceptionally strong in providing tools for creation and maintenance of the knowledgebase. The power of its environment for providing such features, utilizing the bit-map display, menus, the mouse, *etc.*, is unrivaled. The author uses a full-feature editor to create text material to be integrated into the knowledgebase. Graphics functions or particular devices (such as videodisc) can also be provided for specific training areas. A variety of tools provide views of the resulting structure and show the implications for the learner. In addition, semi-automatic tools are used to convert pre-existing, machine-readable text of the subject matter into THOUGHTSTICKER data files.

Conventional computer-aided instruction systems provide authoring tools that are basically passive so far as the content of the presentation to the learner is concerned. THOUGHTSTICKER provides a number of active tools that facilitate the authoring process:

- THOUGHTSTICKER suggests key topics by which to represent the explanation in the knowledgebase; it searches the text as provided by the author, looking for variations and similar terms in the current author's, as well as other authors', knowledgebase.

- THOUGHTSTICKER checks the existing knowledgebase of all authors and reports how its contents relate to the new statement. It suggests how the statements might be related (identical, containing, contained, *etc.*).

- In certain cases THOUGHTSTICKER can detect a possible conflict between statements (technically speaking, it does this not by the semantics of the text but the structures of the knowledgebase the text expresses; THOUGHTSTICKER does not yet contain natural language processing). The system offers a series of methods to resolve the conflict depending on the structures: statements may be declared "not accepted", they may be merged with others, distinctions may be added, *etc.*

- In all cases the author's input is tagged to that author and other key parameters such as time of entry. Some THOUGHTSTICKER user interfaces provide the identity of the author at all times; others display it when the distinction is required. Any authors' denials of a statement are also so tagged, and hence many-valued disagreement and consensus may be stored. (A denial is the modification of a statement relative to a user, as to whether that user accepts the statement as valid or not. This applies the user's own statement as well as to the statements of other users.) In this

way, local extension or modification of the contents of the knowledgebase is easily achieved while still preserving the original.

- To stimulate the author to add further structure and material to the knowledgebase, THOUGHTSTICKER will propose new structures which do not yet exist and which, if instated by the author, will not conflict with existing structures. This process can be focused by having the author indicate areas to extend or areas to avoid. Alternatively, THOUGHT-STICKER can suggest areas that are "thin" compared to others; in this way the author is encouraged to achieve a uniform level of detail.

Unique to THOUGHTSTICKER, the combination of these features make the process of creating the subject matter much more efficient. In addition, multiple authors, possibly at different sites, can contribute to the same knowledgebase without interfering with each other. The original knowledgebase can be augmented and tailored to differing needs at different locations.

## 3.4 Related Software Systems

THOUGHTSTICKER, Lp and dynamic graphics displays of knowledge representations have implications for the domain of software systems as they are now presented in both commercial systems and research programmes. The work of this dissertation presents innovations in these areas, briefly discussed in the following sections.

### 3.4.1 Database Management Systems

The concept of a database is a simple one: to store and index data in a form for swift and convenient retrieval and update.

Approaches to database management come in various forms; the most flexible of which is the most complex to implement but also the most general and most useful. These "relational database" concepts find mature implementations in modern, commercial database programs available on computers from large to small. Their power derives from a complete flexibility in how the data is indexed: truly relational systems can be indexed on any entry in the database. This corresponds to THOUGHT-STICKER's capability for every topic object to be accessible directly, and for any relations that topics exist in to be used as a means to move from relation to relation. Details of implementation aside, THOUGHTSTICKER is functionally a complete relational database.

- Consider that database connections are arbitrary and unconstrained; THOUGHTSTICKER provides structures that model cognitive relationships. Hence, the result may be considered a "knowledge representation" or "knowledgebase" rather than a mere database. Of course in both cases the data contained must be interpreted by a human to make it alive with meaning and become true "information"; however in the case of THOUGHTSTICKER, the structure reflects contextual relationships that are valid in the construct of the creator or author of the structure.

For the user, the nature of the two systems (relational databases and THOUGHTSTICKER) is completely different. Database require that the statement by the user be a "well-formed expression" which can be syntactically parsed and interpreted logically. For example,

(AND (OR (subject = cybernetics) (subject = protologics)) (type = thesis))

would retrieve all records on the subject cybernetics or protologics which are a thesis. Modern systems often employ pseudo natural language input schemes, whereby the same search could be performed by typing, literally,

Show all records of the subject cybernetics or protologics, that are a thesis.

THOUGHTSTICKER in its present forms is not tailored to perform precisely this type of search. However, it can be used in such a mode by two of its mechanisms:

- It can use features of the entries, such as type of entry and contents of the relation to prefer or exclude some entries over others.

- The course of the conversation includes a context of previous requests which are used by THOUGHTSTICKER to determine what data is retrieved.

Note that the relational database search is independent of all previous and future searches; it is without context. THOUGHTSTICKER, by contrast, builds a history of interaction by tracking all requests and modifying subsequent responses. For example, first a request for cybernetics as a topic would recall all such available entries (*i.e.* relations and their models; see Section 5 for full explanations of these terms). A second request for the topics protologics would first provide those relations which overlapped or were close to cybernetics; thereafter, entries that

were individually related to one or the other. Finally, a request for entries on the topic thesis would first retrieve entries that are in cybernetics or protologics.

Thus the retrieval is not a one-shot and without context, but rather an emerging purpose that is created by the history of requests on the user's part. The result is less immediate. (Of course the conventional search patterns could be added as a capability to THOUGHTSTICKER, making it a subset of relational database systems.) However, for application to research where a fixed answer is not sought but rather a picture is to emerge over a series of refined retrievals (which database retrieval usually is) THOUGHTSTICKER holds great promise as a revision to the nature of database management. The free form manner in which statements are added into the database and the lack of restriction on "keys" are substantial improvements.

### 3.4.2 "Thought Processors"

There was a brief flurry of interest in commercial personal computer markets for programs that were erroneously dubbed "thought processors." In fact, each of these were merely word processors with a fixed format for creating outlines. With the appropriate command, a given line in the outline could be expanded to contain sub-lines. The process is fully recursive. The resulting outline would make an excellent basis for writing a full document; hence the claim of "thought processing", which here means instead to help plan the writing process.

THOUGHTSTICKER is rather more like a true thought processor because of its power in cognitive modelling. The structures that result and the process of conflict resolution are a strong partners in the thinking process. In fact, THOUGHTSTICKER is to modelling thought processes

as word processing is to writing. No other commercial or research software can make such a claim.

# 4. Foundations of Conversation Theory

This chapter provides background and the bases of the argument of the thesis. A very brief synopsis of this chapter was the content of the Abstract.

## 4.1 Interaction and Conflict

Conversation Theory is a theory of interaction. The minimum psychological observable is that of an interaction between two distinguishable entities, the distinction of which is made by an observer (Pask 1975b, Pask 1975c). The role of the observer and the interaction are so inextricably linked that they are duals; one does not exist without the other (Pask 1976c, Pask 1980c). Hence, from interaction arises all individuals, all distinctions and therefore all "conceptions." These make up (or "inhabit") the organization of systems as a whole.

The existence of a distinct entity is an observer phenomenon that is consistent with other distinction logics (Varela 1975). The persistence of an entity is the result of a convergent process rather than, for example, the physical existence of a mass (von Foerster 1977).

A range of possible types of interaction arise within and among systems. Trivial interaction is that which is consistent to and meshes smoothly with the existing organization and therefore merely reinforces that organization. Information introduced into a system which is not "novel" is an example of this (Pask 1980b). The crucial case is when the information introduced is novel so far as the system under scrutiny is concerned. Hence, if interaction is to allow for change and evolution of organization, it must perforce consist of occasions where processes (by definition, programs that are executed in one or more processors, Pask

1980b) are not mutually consistent and do not smoothly mesh and where the organization is in danger of change. This is **conflict.** Without conflict, the organization cannot undergo change.

One outcome of conflict can be destruction of the organization. Alternatively, if concepts and individuals are to endure under the influence of conflict, it is necessary that conflict be resolved, with the accompanying persistence of organization albeit a modified one.

## 4.2 The Requirements of Representation

Conversation Theory arose in the context of learning environments where the subject matter to be learned required a representation outside of the human subject matter expert. The independence of knowledge (or more precisely, "knowables") from a knower is an absurdity which is often mooted for the purposes of practical implementation in current digital machines, and for the sake of discourse. Hence it is a simple error to lose this point. Current AI research, of course, is predicated on the possibility of knowables without a knower and the nature of this contradiction is not always acknowledged (as it is in Dreyfuss & Dreyfuss 1986, and Winograd & Flores 1986). CT at all points re-affirms the role of the knower.

In addition, other aspects of this epistemological stance of CT imposes certain requirements on the needed knowledge representation, requirements which AI has generally not benefitted from. Communicability, stability (memory), ambiguity and its resolution are all central to cognition and a knowledge representation based on CT must encompass these issues.

## 4.3 The Rise of Lp: Coherence, Distinction and Process

The needs of a knowledge representation as constrained by CT led Pask to invent the protologic called Lp. The term "Lp" arose in context where CT had already been concerned with descriptions in a language called "L". Pask's work had previously involved a formalism containing "L", a symbol standing for any true language (natural, spoken languages as well as the language of dance, gesture, or signs). The requirement was that L have the capacity for questions and commands as well as statements and possibilities. (Classical mathematics and predicate logic does not; see von Wright, 1963 and more recent echoes in Winograd & Flores 1986.) Because the representation underlying cognition was more primitive than that language (in the sense that all languages could be modeled with a common structure and kinetics), Pask added the "p" subscript, meaning "proto" (meaning "primitive" or "original", as in a substrate). Lp is therefore a substrate or structuralism [*sic*] on which would rest a logic or language to carry the richness of human discourse.

Lp describes the interaction of conceptual entities by providing rules that constrain the interaction of these entities and hence model their evolving organization. These interactions are described at the level of concepts, that is, as the interaction of topics in relations that form conceptions. Lp contains injunctions as to how topics can and may interact, including how they may conflict and how their conflict may be resolved.

In its pure form, Lp is a logic of process as well as coherence and distinction:

- Process, in that all entities are the result of interactions, where an entity is that which is stable and recognizable.

- Distinction, in that there arise in the course of the interaction of processes, entities that did not exist before and that are distinguishable from one another by further processes.

- And coherence, in that conceptual entities "cohere" together: their dynamics are such that their process interaction creates stabilities, themselves conceptual extensions of the original.

Due to their characteristics (such as their kinetics, leading to their stability) Lp entities imply models for memory, uncertainty and innovation.

## 4.4 The Distinction of Micro and Macro

The attribution of a term such as micro or macro is made by an observer relative to some purpose. In the context of software simulation, it refers to the "grain" at which elements are chosen as primitive, and the relations between elements are simulated by procedures which related them.

To choose a level of description and to name it "macro" is equivalent to stating that there will exist some elements of the database which will be considered indivisible atoms and processes below a certain level will be asserted rather than acted upon. In the present case of THOUGHTSTICKER software as described below, the topics of Lp will be considered as atoms, and their relationship will be asserted to be dynamic **but represented as a static structure.** This is not a condemnation; it is nothing more than a proper declaration of the status of the database elements, and it serves to clarify the observer's intentions in the nomenclature of declaring it to be "macro."

For some purposes, such as tutorial representations of subject matter as

detailed below, such a "macro" description of topics is sufficient, because the relationship among topics is to be activated by the user, and the mere existence of their relation is sufficient from the perspective of the software.

For other purposes, this level of grain of the simulation may not be sufficient. In particular, it cannot represent the true implications of CT as a model of the dynamics of mentation. It requires a process interpretation of the structures of Lp. This can be achieved by an increasing series of more detailed simulations. The first of this series retains topics as atoms, but provides a process relationship between them (this being the main topic of the remainder of the thesis). A second would be to break the topics down into sub-components, thereby exposing their "internal" structure to scrutiny. This is unnecessary for demonstration of the thesis and is not explored further. However it is appropriate to comment that such an extension of the simulation would be necessary to provide additional evidence in support of the more subtle implications of Lp, in its power to model generalizations of concepts and their creation in abduction.

## 4.5 Static, Macro Representations of Lp

All previous software programs based in some way upon Lp operations have used a description of Lp that encompasses coherence and distinction only. The level of description of these programs has been that of the topics (considered as indivisible atoms); and a level "higher" than the topics themselves, namely, a level of topic relations.

The topics are represented as static elements in a database; they exist not by nature of a process which is executed but rather because of a configuration of 0/1, binary data in a static software structure. Similarly

relations are static aggregates of tokens associating (either by means of pointer structures or common names) the topics they relate. It may be tempting to consider that in order for these entities to be used by the digital machine, a "process" in the form of a program is executed by the digital machine to access them, and that this is sufficient to achieve "process interaction." However, this misses the crucial point that the topics and their relations in a true Lp processor are embodied **because they are executed** as processes, and their attributes and interactions arise from execution; not because they are being accessed as a static token. The qualities of the entities are the result of execution and not simple reference to a list of static attributes. Process interaction can only be simulated in a serial digital machine by pairwise checking; in a true Lp processor, the medium in which the processes are executed (here unspecified) also affords the means for their interaction.

In existing software implementation of Lp, conflict is detected by a simple counting and comparison scheme. The software makes reference to the static data structures and conditions for conflict (described in Section 5.4 on THOUGHTSTICKER) are calculated.

It is important to realize that this calculation is performed by a program that has available to it all necessary information of the organization of the system as a whole. It is a privileged position which is akin to a global or "god-like" view. It is therefore a position taken by a process that is independent of the system itself (in the sense that an observer is outside the system). Because it is independent, the calculation in this form cannot be performed in this way by the system itself.

Because of this view, the level at which the dynamic interaction of the topics is simulated, is here called "macro." For restricted applications of Lp, such as in a knowledge representation scheme for training, this may be sufficient.

## 4.6 Deficiencies of the Macro

However, the macro position has two deficiencies: the fundamental tenet of Conversation Theory, that of true **process**, is missed; and, conflict does not arise internal to the system, but rather is computed external of [*sic*] the system; that is, macroscopically. It may be seductive to say that the existence of conflict can be denigrated and trivialized to a mere artifact of the level of description and its historical origins in subject matter representation. However the rise of conflict **within** systems must be recognized for its power to model the initiation of distinctions, and hence as a powerful engine for innovation arising within the organization of a system.

Without a process component, there is no "available energy" for the system, and further mechanisms would need to be hypothesized. With process, the entire theory holds together in a consistent manner.

## 4.7 Hypothesis: Theory Confirmation in Micro Simulation

All theories consist of descriptions in a language. A description may imply or produce a further description which in science is often called a "result" of the theory. More precisely, such further descriptions are hypotheses or hypothetical statements that are deduced from the body of the theory. A hypothesis or "theoretical result" is usually compared to observations of some environment and when correlations exist the theory is, in some part, "confirmed."

The hypothesis put forth in this dissertation is that a low-level description of Lp, that of an internal and microscopic level in which topics are influenced by "forces" that are exerted by the topology of the conceptual space, would, in its activation as a dynamic process of appropriate

dimension, produce as a result (and hence provide a confirmation of) the macroscopically-observed behavior of the system manifest as conflict and resolution of conflict. The resulting implementation would reify a system modeled in Lp by producing a system whose topological space was constrained by the interaction of its entities.

This has some similarities to the work in new "quantum computability" (Deutsch 1985), which is another revision of "classical" computational theory (*i.e.* that attributed to Turing). There as here the desire is to achieve certain classes of computation which otherwise would not be possible; in particular, computation which would not be possible in any "Turing architecture" consisting of a finite state machine and a tape (memory). This position is in sharp contrast to the historical view of Turing computability as sufficient for any class of finite computation, including that of brain (for an excellent discussion of the interactions between these views see Lettvin 1985). Since the digital computer is based on the Turing model, it was considered just a matter of engineering before computers were smart like humans. The revision requires new hardware architectures.

## 4.8 Dynamic, Micro Representation of Lp

To return to the original intentions of Lp as founded on process, a new software model must be put forward to reify Lp structures. Unfortunately there are fundamental limitations presented by present-day serial, digital machines and a true Lp embodiment must await new architectures (which are beginning to appear, see Section 7.3.4). However the basis for a new approach can be set out now, and simulated in current hardware. The software written for this dissertation, although a simulation, restores the process component to Lp embodiments and provides a direction for future work in fully concurrent machines.

In brief (as the details will be presented in Chapter 6), the individual entities that exist within an Lp structure exist due to the execution of a process rather than their existence in a static database. This can be simulated within serial machines if the interactions (relations) between entities (topics) are expressed as a continuous computation of relationships within a topological space. These relationships are represented to the observer as relative positions on a display screen.

The topics themselves are atomic units and not processes whose execution result in stable (but dynamic) entities. For the purposes of practical implementations, some level of "atom" must be chosen to begin the simulation. But while this is the case, their relations as manifest in a graphical display exist due to the interactions of processes. These processes individually are the action and interaction of each entity within the organization of the system under execution.

Interpreted graphically in this way, topics compute their positions relative to their neighbors in relations that they share. The computation is performed in accordance with Lp rules. The resulting positions, which may or may not be stable, represent the conceptual relationships of the topics relative to each other.

Within the simulation of the micro interactions of Lp, macro features of CT should emerge. For example, for certain initial configurations ambiguity or contradiction should be detected. This prediction is confirmed, as will be shown in Section 6.5.

The next major Section presents a detailed view of THOUGHTSTICKER software at the macro level, which is a necessary precursor to discussion of the micro of Lp and results.

# 5. Conversation Theory Software

## 5.1 The Birth of "THOUGHTSTICKER"

As noted in Section 3.3.2, THOUGHTSTICKER was invented by Pask and collaborators at System Research Ltd in the late 1970s (Pask 1976a). Its development was more or less coincident with the development of Lp in that the creation of THOUGHTSTICKER (as a software manifestation) both fed on and was fed by development of Lp (its formal and notational basis). However, and as already noted, to be a full-blown logic for CT, Lp required a bifurcation principle (described below). Pask has stated that this was available only after the enquiries of Vittorio Midoro about the notation of analogy and distributive coherence (an overlap of a single topic in more than one relation) co-existing in the same diagram. Pask had already realized that some principle was needed to explain how new structures arise from computation performed from within a system. This, along with the principles of conservation, duality and complementarity already formulated, would make CT a complete, "scientific" theory. Midoro's enquiry led to the simple and elegant bifurcation principle that shows how distinctions and hence new structures arise from within an organization. Midoro was at the University of Genoa at the time, and hence Pask has called the bifurcation rule the "Rule of Genoa."

It will be demonstrated that THOUGHTSTICKER in all of its software forms represents the embodiment of Lp at a macro level, an argument that is one foundation of this thesis. To further clarify what is meant by this and to provide necessary background a detailed description of THOUGHTSTICKER follows.

### 5.1.1 Raison d'Etre of THOUGHTSTICKER

The background to the THOUGHTSTICKER system may be seen as two, concurrent, threads:

1. The development of Conversation Theory as a scientific and psychological model for knowledge and beliefs; and
2. To "compute" knowledge structures inside of presently-available digital machines, a goal that is analogous to the attempts of AI.

This chapter focuses on the former, although some comments on the latter are necessary.

### 5.1.2 Represent What?

Both AI and cybernetics have encountered the same dilemma, albeit from quite different paths:

- What is knowledge that it may be represented in a concrete structure; and
- What is a representation that it may reflect the process of knowing?

Without question the issues here are very deep and are properly treated in other places, the literature of philosophy being one (in a monograph particularly focusing on CT and Lp, see also Nicoll 1985).

In the context of the use of computers in human decision making situations, it can be shown why the issue arises at all by the aid of the

following parable: To help humans to perform calculations such as check-book balancing, word processing and orbits of satellites, the computer must manipulate with facility the elements of these domains, such as numbers, representations of text, calculations under specified equations, and so on. Without this capability, the computer would be useless for these tasks.

Similarly, for the computer system to provide any help, support, advice, what have you, in the "thinking process" (alias the "decision making" process) it must manipulate with facility the elements of the domain: the knowledge of the user. This presumes, not unreasonably, that the computer is to calculate a domain beyond mere numbers and measures. The domain becomes the non-quantitative, non-specific and often inchoate world of beliefs, conceptions and impressions. (See comments in Section 3.4.2 concerning so-called "thought processors.")

### 5.1.3 Attempts at Knowledge Representation: "Expert Systems"

The goals of knowledge representation are easily said, but not so easily done. The 25 year history of AI has attempted to deal with these issues from the "bottom up": starting from the computer technology and a reductionist view of mental processes. Some consider that the process has borne fruit (Michie 1982; Feigenbaum & McCorduck 1983) while even the most impressive of so-called "expert-systems" are limited in the extreme (Duda & Shortcliffe 1983).

The "expert system" paradigm is one which considers that the "knowledge" of experts may be captured by a manual process, and converted into a form computable by present-day computers. This manual conversion is performed by a "knowledge engineer" who codes the

"expertise" into rules which are easily calculated over by the digital engine. The tribulations and disadvantages of such a presumptive approach have been discussed elsewhere, in Pangaro & Nicoll 1983.

For our purposes here, it is sufficient to point out that it may be possible to divide the global problem of the reification of knowledge into two stages:

1. Capturing a **representation** which is useful to the user and to others but which cannot be computed over by the digital engine, that is, the computer does not **know**; and

2. Elaborating the structure of representation so that the digital engine may itself perform (*e.g.* **decide**) in a manner which reflects somewhat the form as well as the content of the original human thinker. Humberto Maturana has made the point (Maturana 1986) that going the route of representation separate from ontology is a fundamental misunderstanding of the nature of knowing and any attempts in that direction are doomed. His position is beyond the compass of this discussion, as the center of this thesis are the issues of demonstration and confirmation; hence representation is desired.

Expert systems and AI in general attempt the second, and more difficult stage, first. The goal of pragmatic research programs (Sheppard 1981; Pask 1981) is the former, with a clear plan of extension into the latter, as techniques **and technology** catch up to the more demanding requirements of "machine intelligence."

This claim of capability (for it is only a claim until demonstrated in working systems) is based on the substantial theoretical and experimental work which Conversation Theory represents as embodied, within its limitations, in the software system called THOUGHTSTICKER.

## 5.2 The origins of THOUGHTSTICKER

In some sense it is impossible to pinpoint a "first" implementation of the concepts behind THOUGHTSTICKER as it is now discussed. Pask and his associates produced many machines from the middle 1950s to the middle 1970s, each of them contributing important ideas to Conversation Theory and its fruition in THOUGHTSTICKER. In the late 1960s and early 1970s, a few machines were made that were clear predecessors to THOUGHTSTICKER. These were the CASTE and EXTEND systems (Pask 1975c). Each had an electro-mechanical interface manipulated by the human subject, connected to software programs for purposes of recording data and performing some calculations most conveniently done in software.

### 5.2.1 The Demands of Course Assembly

The very need for a system to represent the structure of knowables grew out of the problem of representing subject matter for environments for learning. The ultimate structure of the representations grew out of the epistemological foundation of cybernetics, in the form of Conversation Theory itself.

It is interesting to note that the need for a representation of subject matter for teaching came before the theory of conversations or its strict calculus of knowledge representation. Once CASTE was mature as a "Course Assembly and Tutorial Environment", THOUGHTSTICKER

was conceived as a software aid to assembling the structures which would "hold" the subject matter for tutorial purposes.

The distinctions between CASTE and THOUGHTSTICKER are a source of confusion since recent usage of these terms has tended to imply different implementations in different hardware but both based on Conversation Theory and Lp. For the record, recent uses of the name THOUGHTSTICKER emphasize the knowledge representation functions and the name CASTE emphasizes the tutorial heuristics. However, any THOUGHTSTICKER demonstration usually includes some CASTE functions for purposes of practical use and demonstration, while CASTE operates on the structures produced by THOUGHTSTICKER. Hence either term implies the other and neither can be independent.

EXTEND was a related software program which allowed for the user (whether in the role of "teacher" or "learner") to extend the subject matter representation.

Thus it was subsequent to CASTE, EXTEND, and even THOUGHT-STICKER that, with the invention of a bifurcation principle, CT produced what Pask would consider a complete, scientific theory capable of encompassing, minimally, the domain of epistemology.

## 5.2.2 THOUGHTSTICKER Defined

A precise distinction between Lp, Lp software and THOUGHTSTICKER was originated by C Sheppard and R (Dik) Gregory of Admiralty Research Establishment (ARE), Teddington, UK. "THOUGHT-STICKER" indicates a user interface written in software and connected to a software embodiment of Lp structures and processes ("Lp software"), within the constraints of present digital technology, which constraints are very great compared to the intention behind the formal protolanguage

itself ("Lp"). Multi-process, concurrent, conflict-ridden as well as conflict-free computation are a few of the gross omissions inherent in any present-day THOUGHTSTICKER. Even the proposals of modern AI for non-von Neumann, many-processor digital hardware is not capable of the proper processing that is required for Lp. Some of these points will be encountered more fully in the ensuing argument, below.

Even given these real restrictions, the potential benefits of a THOUGHTSTICKER are very great as applied in a direct way to database construction and retrieval, computer-aided instruction, and a variety of tasks that involve multiple-authors and/or multiple locations. Its practical implications for decision support and machine intelligence are only implied and as yet unexplored.

### 5.2.3 THOUGHTSTICKER in its current forms

The specific history of THOUGHTSTICKER implementations is offered in the Appendix C as it is tangent to the main thesis. For our purposes here it is sufficient to describe current implementations.

At present there are two major embodiments of THOUGHTSTICKER in software available for examination.

**Microcomputer BASIC Versions**

There are two versions running in the Apple II microcomputer with additional hardware boards. One, called Apple CASTE, was developed largely for the Admiralty Research Establishment, UK, with some modules and features added for the US Army Research Institute (ARI), by PANGARO Incorporated. Another is called C/CASTE, based on the original code of Apple CASTE and developed for the US Army Research Institute at Concordia University under the direction of Pask. The systems

have as their strengths that they are self-contained in available and inexpensive hardware, and are well debugged and documented. Their limits are in the size of the database they may comfortably contain and the restricted set of Lp operations they perform.

Both contain the basic Lp operations (up to but not including condense/expand and generalization). Apple CASTE emphasizes the authoring and presentation of text models for Lp entities, although a simple Apple graphics module can be used. In contrast, C/CASTE emphasizes the multi-display presentation of tutorial material including computer-controlled slides of the subject matter and maps of the knowledge representations.

Both use CASTE in their names, relating them to the Course Assembly System and Tutorial Environment, the system that preceded THOUGHTSTICKER and Lp, because the focus of their use is the tutorial application of CT.

### Symbolics LISP Versions

This is an extended version of Lp operations, including simple generalization, bifurcation, and extended conflict resolution, running on the Symbolics LISP Machine. The THOUGHTSTICKER code is manifest in a number of forms on the Symbolics, including a series of user interaction frames for studying the evolution of the knowledge representation; a "naive" interface for users without knowledge of CT; and the Expertise Tutor, used to teach a naval command and control task. The power for the system is very great due to the power of the environment of the Symbolics, its speed, size and efficiency of experimentation and implementation. The implementation surpasses all previous versions in raw functionality and capability.

Further details of the history of software development of THOUGHTSTICKER, including its successful integration into a complete CT system of discourse (the Expertise Tutor), are found in Appendix C.

The explanations below will use the Symbolics version for its examples, although the particular details of screens and menu functions are minimized as they are specific to this implementation; the concepts are however general in the context of CT.

### 5.2.4 Lp at the Macro Level

As defined above in Section 5.2.2 and in distinction to Lp software and Lp itself, THOUGHTSTICKER is a software program which provides access to a set of Lp functions in software. The formal description of its functions is that of Lp, which in turn is the dual of CT, a macro theory of conversations from whence it arises. Although it cannot be a complete implementation of Lp (for both technical and theoretical reasons) THOUGHTSTICKER is used below to explain the operations of Lp. Details of the full operation of the software can be found in Pangaro *et al* 1985, attached.

### 5.2.5 Uses of THOUGHTSTICKER

To elicit knowledge from users, software such as THOUGHTSTICKER may operate in one of two modes:

1. In the background, accepting input from a domain (as from a simulation called HUNKS for the ARE , or from the Team Decision System (TDS) as developed for ARI); or

2. Directly, with the user and the elicitation software engaged in the one-on-one interaction.

The second is the model used for the following description as it is the more general case. The discussion focuses on THOUGHTSTICKER as an engine for receiving and representing knowledge without direct concern for the ultimate structure and its kinetics, the central issue of this thesis. However it is essential to present in detail the meaning and interpretation of the structures at the macro level (to the "user" and his or her "psychology") and from that presentation make the case for the correctness of interpretation of the theory at the micro level (to the "topics" and their atomic interactions).

The frames used as examples below are from the "research" version of THOUGHTSTICKER, that is a set of interaction windows constructed in 1983 and 1984 whose purpose is to allow the user to easily explore the knowledge representation scheme behind THOUGHTSTICKER. A simpler, "Naive THOUGHTSTICKER" is also available for users not wishing to be exposed to the internal issues of the scheme.

## 5.3 Making Statements

In a one-on-one interaction with THOUGHTSTICKER, it is the user's responsibility to take initiative in making assertions which THOUGHT-STICKER endeavors to represent in its internal structures. It is THOUGHTSTICKER's responsibility to conform the user's actions to its internal requirements and to engage the user in a dialogue when there is conflict or disagreement between the user and THOUGHTSTICKER. There are additional features of THOUGHTSTICKER that provide stimulation to the user in hopes of initiating novel assertions.

The primary way in which the user adds to THOUGHTSTICKER's

internal knowledge representation is by making text statements. Further transactions are required to characterize the meaning within these statements. All this is accomplished by display screens, menus of functions, and the ability to point to sections of the text to indicate words and phrases within the user text.

Figure 1 shows a Write Watcher "window", as a software screen of the Symbolics is called. It consists of "panes", each of which is surrounded by a border and contains elements such as text and symbols. The user in the role of "author" begins by typing statements at the keyboard, which are entered into middle pane as if into a word processor which is contained inside of THOUGHTSTICKER. The usual English language conventions of grammar and punctuation are followed to the discretion of the user.

The first important distinction to make about interaction with THOUGHTSTICKER is that the system performs no semantic processing. This means that details of sentence structure and grammar are ignored; the text in its entirety is recorded by THOUGHTSTICKER for later retrieval. This is not to say that a natural language interface would not be to advantage at the THOUGHTSTICKER interface; it is an elaboration which is hoped for in future development.

Let us presume that the user has made a statement which reflects his or her belief about a particular subject. In this case, the statement is "To represent knowledge is the goal of artificial intelligence programming." The result of typing the statement is seen in the lower pane. Certain words or phrases appear in that pane in bold type. This indicates that users have previously distinguished these words and phrases to THOUGHTSTICKER as significant and to be noted whenever they appear. These are the topics of CT.

### 5.3.1 Models, Topics and Relations

It is important to differentiate the various elements of the authoring process in terms of the strict definitions of CT:

1. The text of sentences which are typed by the author are **models.** They are not modified by the system itself but are "executed" (that is, printed onto the screen) as a manifestation of the object they model, whether a topic or entailment (relation).

2. Models are associated with **entailments.** An entailment is a grouping of a particular type, for example, a coherence or an analogy.

3. The elements of entailments are **topics.** These are the exteriorized elements of **concepts,** which are themselves clusters of processes. Topics are the public elements of concepts (whether shared among different individuals or merely exteriorized into the THOUGHTSTICKER interface). In the present THOUGHTSTICKER, topics are represented by words or phrases.

The implications of entailment will be discussed below but to keep this exposition reasonably brief, exceptions and qualifications to statements will be minimized. Some implications worth noting are: models may very well (and in certain cases, **should**) be graphics or sound; topics are not the same as words or phrases but are efficiently represented so; models need not contain identical words/phrases as the topics of the entailment they model.

THOUGHTSTICKER can extract from the text of the model topics' words and phrases which it already has noted, as Figure 1 shows. Not all topics will have necessarily been asserted to THOUGHTSTICKER, and new topics can be added manually by the author. This is accomplished by pointing at the words or phrases in the lower pane. THOUGHTSTICKER checks to see whether the text is close to previous topics (for example, is the new topic "goal" similar to previous topics, such as "goals" or "goal structures"). The user may indicate that these new topics are intended to be the same as earlier ones, or to be kept distinct.

The resulting topics are displayed in a separate pane (second from the top) labeled "Topics."

### 5.3.2 Instating Entailments

Up to this point, the author has made a statement into THOUGHT-STICKER in the form of a few sentences of text. This is to serve as a model of an entailment involving a set of topics, which the author has also indicated to the system. The next step is to integrate this new knowledge into the pre-existing models and entailments which THOUGHTSTICKER holds.

### 5.3.3 Coherence

As noted, commands to THOUGHTSTICKER are made by choosing items on a menu. The label on the menu choice in Figure 1 is "Instate"; clicking here will provide further choices, to instate the utterance into the database as a coherence, analogy, or topic model. Referring to the term from CT, a **coherence** is an entailment between topics which requires, first, that each topic in the entailment "make sense" with its neighbors in the entailment. Thus, the meaning of **knowledge** must be derivable from

Artificial Intelligence and goal; recall that the purpose of the model is to represent this meaning.

However, within a coherence every topic must be supported and "producible" from all of its neighbors in the relation; hence, artificial intelligence may be explained as something which has as its goal the representation of knowledge. Equisignificantly, a goal is seen to be made from the entailment of knowledge and artificial intelligence; the topic goal is their entailment.

This requirement for coherence between topics distinguishes THOUGHTSTICKER from all other knowledge representation software available in the field of AI.

There is the issue of choosing how to model a given utterance, namely, how it should be instated. The requirement for mutually-producible topics, just described, is the minimum test for coherence. Lesser conditions can qualify as analogy. Topic models are used if they require no further breakdown of detail, that is, if the topic itself is an "atom" of the subject matter. (The "Naive" modes of THOUGHTSTICKER provide a semi-automated approach to this, where questions about the utterance are used by the system to guide the user through considering how to best model the utterance in the knowledgebase.)

### 5.3.4 Subjectivity of Statements

A few observations are in order at this stage. One may argue as to whether one of the topics should really be "artificial intelligence programming", rather than simply programming. This, as everything about the process which the author undertakes, is a matter of opinion. Nothing about the representation is true, immutable, or correct. It is merely the belief of the author, in the context in which he or she is

making statements. Thus, THOUGHTSTICKER is a system for reflecting subjective assertions, namely, beliefs.

One may also argue about whether "goal" is really produced from the remaining topics. Again, this is a matter of the opinion of the author, but unless the sense of production can be comprehended by users other than the author, the results will not be so useful. THOUGHTSTICKER contains specific features, described below, which endeavor to make the knowledge structure as comprehensible as possible (but always, of course, within the limits of the communication skills of the author). Without the concept of coherence, however, THOUGHTSTICKER is merely a keyword database retrieval system. With coherence, it is a unique system for testing agreement between individuals.

As will be seen in later chapters, coherence may be seen as a set of forces imposed by and acting upon topics. In THOUGHTSTICKER, these forces are computed by external fiat (see next section); however CT requires that they be dynamic forces acting within the relations themselves.

## 5.4 Contradiction Checking

THOUGHTSTICKER must insure that new assertions are "consistent" with old ones, in order to maintain the coherence of the knowledge representation as a whole. THOUGHTSTICKER uses the requirement for coherent entailment as a basis for evaluating the internal consistency of the knowledge representation, as follows.

Upon the user's injunction of "Instate", the system compares the proposed entailment against all previous entailments. Because THOUGHTSTICKER does no semantic processing, all evaluation is done purely structurally, by examining the topics in their entailments in

the entire knowledge representation. In essence, THOUGHTSTICKER searches for overlaps of the proposed entailment with past entailments. If THOUGHTSTICKER determines a potential contradiction, the user is warned as shown in Figure 2. A new menu has popped-up on the screen, stating that the proposed entailment has conflicts in the mesh and offering among other menu choices "Try to resolve conflicts." The other menu choices refer to alternative interpretations of the same text model. These are available if the user wishes to back away from asserting a complete coherence. However by far the most interesting case is that of conflict resolution, described in the following sub-sections.

### 5.4.1 Cases of Contradiction

The situations that THOUGHTSTICKER can detect are from the following cases:

1. There is no overlap of topics: the new entailment is completely unrelated to existing ones, and indeed contains topics which appear no where else. The proposed entailment cannot be contradictory, but is unrelated. As, for example in the present case, "**Cybernetics** is the **epistemology** of **science**." (Intended topics are **bold**.)

2. There is overlap on one topic only: in CT, this is called **distributive entailment**, because the meaning of the overlapping topic is distributed across more than one entailment. There is no structural contradiction and hence the proposed entailment may be accepted. For example,

"**Artificial Intelligence as a field was established in the 1950s by McCarthy, Minsky and others.**"

3. There is overlap on all but one topic; for example, there are some identical topics present in both the proposed entailment and a previous entailment; and two further, but different, topics, one in the proposed entailment and one in the previous entailment. For example, "The goal of **Artificial Intelligence** is to **make computers smart** like people." Within CT this is the classic case of contradiction and requires some explanation, below.

4. There is a common subset of topic names between the proposed entailment and a previous one. Is the entailment with fewer topics a "conceptual subset" as well, in that it is entirely contained in the larger one? "**Artificial Intelligence** involves the embodiment of **knowledge** into computers for the purpose of **makeing computers smart** like people..." would be an example of this case.

5. There is complete overlap: all topics in the proposed entailment are contained identically in a previous entailment. Since THOUGHTSTICKER (as noted) performs no semantic processing, the question arises: Are the entailments truly identical? Or, was a new, different entailment intended by the author? For example, "The goal

of **Artificial Intelligence** is to program machines to behave as if they contained the knowledge of human beings."

Of course, any of the cases of contradiction may exist with more than one previous entailment.

In all of the cases cited, the procedure is one of comparison with past entailments and (if necessary or desired) a resolution of the conflict based on the author's intention. It is conceivable that any keyword retrieval system could point out the condition of keywords in common, although without CT as an underpinning, there would be no reason to draw any implications from the particular structure in each case.
THOUGHTSTICKER can provide specific aid in interpreting and resolving the contradiction, as exemplified below by the particular and perhaps most interesting case of classic contradiction, where all but one topic in both entailments overlap.

### 5.4.2 Resolution of Conflict

Returning to the same statement example, THOUGHTSTICKER had detected a condition of possible conflict between an existing entailment and the new statement just made by the author. It is now up to the user and THOUGHTSTICKER to modify the structure if resolution of conflict is desired.

THOUGHTSTICKER responds to the user injunction "Try to resolve conflicts", shown in Figure 2, by offering a new window called the Resolver, shown in Figure 3. This window shows the proposed entailment on the middle left, with the previous entailment that it conflicts with, on the middle right. The "Shared topics" is shown in a pane in the upper middle of the window, "**Artificial Intelligence**" and

"knowledge", are in both entailments. "Goals" is present only in the proposed entailment (left side) and "data structure" is present only in the previous entailment (right side). The symmetric menu choices on each side represent various procedures for the author to follow to resolve the conflict; for example, to "Deny" one model or the other. Other functions are aids to the user, for example "Undo" returns to a previous state, and "Describe" gives relevant details of the structure of the nearby database. As before, details of function are available in Pangaro *et al* (1985).

As noted, the significance of the detection of conflict comes from the implication of coherence: each topic is producible from the remainder of topics in the same entailment. The present situation implies that the same topics (the ones which overlap in both the proposed and previous entailment) may produce either of two topics, thus:

- **Artificial Intelligence** and **knowledge** produce **goals**; and
- **Artificial Intelligence** and **knowledge** produce **data structure.**

Which is it? Either/Both? Each, but with qualification? Let us examine the possible resolutions in detail:

1. The non-overlapping topics are really the same topic: in this case, **goals** and **data structure** were perhaps originally intended by the author to have the same meaning. Here, it is not the case, although one can easily imagine an author inadvertently using two different names for topics (for example, **goals** and **purposes**, or **data structure** and **internal representation**) while meaning the same thing.

2. The two entailments should really be merged into one, relating all the topics of both entailments. In this case, the result might be a model such as: "The goal of **Artificial Intelligence** is to capture **knowledge** in software **data structures.**" Let us suppose for our purposes here that this is not sufficient for the author, as the previous entailment was making a slightly different point.

3. One or more of the overlapping topics are not really a single topic but are two (or more) as related by analogy. For example, **Artificial Intelligence** is, in the proposed entailment, a field of programming; whereas in the previous entailment, the statement is about the proponents of Artificial Intelligence, the individuals themselves. A fine distinction, to be sure, but one which must be accommodated within any knowledge representation scheme. THOUGHTSTICKER would accommodate this by splitting **Artificial Intelligence** into **Artificial Intelligence programming** and **Artificial Intelligence proponents**, as joined by an analogy entailment, **Artificial Intelligence.** The overlap of topics would now merely be a distributive entailment and the structure would be coherent.

4. Or, the author's intention is yet more subtle than any of the above, and it was only through the author's process of

semantic comparison that the real intention is clear. In our example, this requires editing of both of the existing models and a modification of the topics in the corresponding entailments. Choosing "Modify" results in the appearance, as in Figure 4, of a new pane which is used to modify the proposed entailment. Figure 5 shows the same for the previous entailment. These smaller panes are analogous to the original Write Watcher window and allow text editing, choosing of topics, as well as the ability to examine previous uses of topics in other entailments.

The outcome, where the text models and the topics in the entailment have been changed, is one where there is no longer conflict within the entailments we have been dealing with. This is indicated in Figure 6 by the new menu choice "Local Resolution" in the top middle of the window, which when chosen accepts the two entailments in their modified form. However, the resolution is only **local**; this means that **other** difficulties may exist between the **new** entailments and previous ones. The procedure is thus recursive.

Contradiction checking by THOUGHTSTICKER does not in itself result in a definite judgment that resolution is mandatory. The judgement is entirely the user's, and the user may decide to perform a resolution or not, depending on the purpose of the resulting knowledge representation.

### 5.4.3 To Resolve

In art, contradiction and its dual, ambiguity, are often used for conscious effect. For psychological modelling, also, the existence of contradictory

structure may be appropriate. The important idea is that THOUGHTSTICKER can represent what the author wishes, and the flexibility to provide for any belief is one of its strengths. For tutorial purposes (and for distributed planning and decision making), surely a self-consistent structure will be appropriate.

## 5.5 Analogy

Like coherence, analogy is a type of relation within CT. A complete discussion of analogy would require space far beyond what is practical here, as it is the foundation of coherent relationships between topics and the basis for condense/expand operations (where the evolution of analogies leads to the creation of independent organizations of structures and possibly to innovation). A synopsis is provided below rather than omit the topic but it is not complete in the implications of analogy to Lp.

### 5.5.1 The Form of Analogy

At a level of modeled structure, analogy consists of a group of topics, a similarity term and one or more difference terms. The similarity term indicates how the topics are similar, while the difference term indicates how they are distinguishable, *i.e.* distinct.

At all places at the interface, THOUGHTSTICKER allows for the user's choice of analogy or coherence in instating a relation. Since current Lp software does not dynamically interrelate coherence to analogies, the contradiction checking is not modified by the existence of analogies; in future this should be the case. Likely forms of resolution could be determined by the software itself using analogical structures, and proposed to the user as candidates. In an automatic Lp processor, each proposal could be executed concurrently with the "richest" paths instated as new structures.

### 5.5.2 The Relation of Analogy and Coherence

Analogy is the most primitive form of relation between topics. Consider that an analogy (at least) relates the (say) two topics that produce a third. If, in addition to that production, one of the two producing topics and the produced topic can also produce the second producing topics, then a further and distinct analogy exists between those topics. The addition of the final production (the other producing plus produced produces the first producing) produces a condition that is recognized as the requirement for coherence. Put another way, the existence of the necessary set of analogies is coherence.

The rise of analogies is the necessary pre-condition to the existence of coherence.

### 5.5.3 Analogy and Distributivity

The distributive case refers to a single topic overlapping in two or more coherences. This is a very common event as a large structure of relations could not easily exist without such a means to overlap relations. This status of distributivity is important for a variety of reasons within Lp.

Consider the topic on which there is an overlap. From one perspective, the topic is an atomic unit that applies to the (let us say in this example) two coherences that it is present in. Put another way, the two relations overlap on the **similarity** of the topic in both relations. However, since the topic is produced in different ways in the two relations, some **differences** must exist that could be extracted out of the two means of producing the topic in the individual cases of the two relations.

Clearly the distributive case contains within it an analogical relation centered on the topic in common to the relations.

Some comments are made in Section 6.4 concerning the role of analogy in the microscopic simulation of Lp.

## 5.6 Adding Coherent Relations: Saturation

The philosophy of THOUGHTSTICKER is to provide the user with feedback which is provocative to the authoring process. Contradiction checking is one such feedback process, which indicates the way in which new statements relate to previous ones. There is a converse situation, in which the system could suggest entailments which are not yet present in the knowledge representation, but which nonetheless would be permissible according to the rules of checking contradiction.

Saturation is an operation whereby THOUGHTSTICKER suggests new entailments to be made. This is analogous to a conversational partner asking for more information about the relation of existing topics in the conversation, but in new combinations. The challenge is to propose new combinations for which the author is likely to want to, and be able to, provide models. The term "saturation" implies that the entailments among topics are being filled in or saturated, making a richly interconnected network of relationships.

In a large domain, the number of combinations of topics is very large and most combinations would not be sensible to use as the basis for new models. Arbitrary combinations chosen by the system would be absolute nonsense nearly every time. The exceptions might be the seeds for innovation, as when very different ideas are juxtaposed, making a new and unforeseen entailment (which is the entire concept behind DeBono's "lateral thinking"). At issue is the efficiency of the entire process, and the likelihood of useful suggestions.

THOUGHTSTICKER must contain additional mechanisms for "focusing" the saturation process to minimize absurd suggestions and to stimulate the author in an efficient manner. Experience has shown that a combination of these techniques results in an effective authoring process.

The first means of focusing the suggestion process is to use contradiction checking to avoid new entailments that would conflict with existing ones. THOUGHTSTICKER produces a possible combination of topics and checks the possible entailment against existing ones. If a conflict exists, the suggestion is discarded and a new possibility is generated combinatorially.

A second means to focus the saturation process is for the author to specify what range of topics to choose from in the composition of new entailments. The author indicates one or a few topics to start from, and requests THOUGHTSTICKER to gather all topics which touch upon those topics in existing entailments. The process may be repeated, reaching further out from the initial topic(s) as far as the author wishes (with the limiting case of the inclusion of every topic of the mesh in new suggestions). This is equivalent to asking the system to make suggestions within a certain area of the knowledge representation, for example, the part dealing with "Artificial Intelligence" and all topics which connect to it.

A third means of focusing the saturation process is to require THOUGHTSTICKER to include certain topic(s) in any new proposal, or, conversely, to avoid using certain combinations of topics. The former is equivalent to specifying a theme around which new entailments are to revolve, for example, all new suggestions are to contain "Artificial Intelligence." The latter is equivalent to specifying that "Artificial Intelligence" and "windows" can be eliminated as a possible combination because it is incongruous, or simply because the author has nothing to say about them together.

Saturation derives its power from coherence and contradiction checking. It is conjectured within CT that the saturation process is constantly at work in the processes of intelligence, connecting and re-connecting concepts as they are generated or integrated from outside information. It is this process which is the foundation of agreement. Without the interrelation of pre-formed concepts with the influence of new concepts, knowledge would be trapped within its own capsules, and perforce could not evolve or even come to exist. The saturation operation of THOUGHTSTICKER mimics this process of mind in a crude fashion to provide a limited but provocative partner in the process of knowledge elicitation.

## 5.7 Tutorial Aids

Presuming that an author (or team of authors) has built up enough models, entailments, and topics to constitute enough subject matter that learning it is worthwhile, THOUGHTSTICKER provides a series of user transactions to make such learning efficient and adapted to the user. These transactions are normally described under CASTE (Course Assembly System and Tutorial Environment), detailed in Pangaro & Harney, 1983.

## 5.8 Implications of THOUGHTSTICKER

At one level, THOUGHTSTICKER is a system for absorbing the utterances of human users and forming structures which reflect the kinetic knowledge of the original user. It is important to stress, however, that this is possible only through the process of agreement.

CT has much to say about the process of agreement and strictly defines it. Informally, it may be considered to be the matching of descriptions and

procedures associated with a particular concept, across participants. THOUGHTSTICKER, as a software embodiment of CT, represents concepts as topics and their entailments. The entailments themselves have models attached, which may be text descriptions (as in the present version), or pictures, or sound.

It is the users' responsibility to perform the matching process across topics. This is done by examining the topic's entailments and giving the entailments meaning *via* its models. In the authoring process, this is manifest by the author checking that the use of a topic name in a new entailment is consistent with previous uses. THOUGHTSTICKER allows for this, as for example in Figure 7, by displaying at the user's request all known names for a topic as well as all known entailments. A full tutorial may also be requested, placing the author in the role of student for the purposes of exploring what the knowledge representation already contains.

The process of agreement is, at present, performed in the mind of the user, but is facilitated by the features of THOUGHTSTICKER. Consider that the ability to form agreement is the heart of human conversation and that THOUGHTSTICKER is one of the first systems for facilitating the process.

## 5.9 Many Authors Conversing

The examples thus far have implied a single author. Forming agreement within an author's knowledge representation is clearly important. The issue becomes much more important when there are many authors, perhaps distributed across many individual systems that are geographically separated. In this situation there is no opportunity to share meanings outside the system itself, and much more responsibility is placed on the interface itself to facilitate agreement.

The situation of the previous section concerns a match between two particular representations for topics: the words or phrases are shared between the two cases, or perhaps they differ only by a difference of singular/plural, or grammatical tense. For example, THOUGHTSTICKER detects the similarity between "coherence" and "coherent." Another case is "language" and "programming language", where there may be clear differences of meaning --- unless of course all uses of "language" in that subject matter mean programming language. This is a transparent example of how uses of terms that are personal to one user or to one subject area may be handled by THOUGHTSTICKER. At any point, the system offers the opportunity to explore how the existing topics are used in their various entailments by picking an option on a choice menu.

A considerably more subtle side of this general problem of agreement over use of terms occurs when different words or phrases are used to represent the same topics of different authors. Here, THOUGHT-STICKER is not capable of evaluating whether such is the case, at least not without a natural language processor. Two extremes may be considered:

1. Where the authors have entirely separate vocabularies, and no two topics are represented by words or phrases that are at all related. This is equivalent to the case of speaking different languages entirely, say, English and Japanese, a situation in which no conversation may occur. In such human situations, some commonality of need or context is maintained as the basis for exchange, and the role of additional modes, such as gesture, facial expression, *etc.*, is

paramount. No system or mechanism is capable of making connections across this gulf.

2. There is some overlap of terminology, but it is by no means complete. THOUGHTSTICKER responds in this situation with the existing mechanism of contradiction checking, and displays the overlaps of related topic words and phrases. This allows the user to interpret the models of the entailments (the text explanations which were authored) and evaluate whether other authors' terms are the same, or at least connected by analogy, to his or her own. In this sense, the "contradiction checking" mechanism at the heart of THOUGHTSTICKER could be better called "agreement checking."

Of course, it is the latter case which always occurs in reality, whether in discourse that is face-to-face or mediated by THOUGHTSTICKER. No two individuals have identical vocabulary and concepts, or they would be the same individual. The current THOUGHTSTICKER has some restriction in mode of expression, namely, restriction to the text which comprises the models and the topics. Nonetheless, mechanisms within THOUGHTSTICKER aid the user in reaching agreement with others' (as well as one's own) elicited knowledge structures.

It is important to note that THOUGHTSTICKER could provide a much richer environment for agreement checking if it contained other types of models; for example, graphics and animation, or sound. The author would construct such models and THOUGHTSTICKER would attach them to

the entailments. This would allow a greater range of interaction for users and conceivably achieve a confidence of agreement not possible through the single mode of text.

## 5.10 Personalized Vocabularies

Once these difficulties of agreement are handled for the case of different user vocabularies for the same or similar topics, the encouragement to maintain a common vocabulary may be relaxed. Users may diverge on opinion but if they "agree to disagree" in the CT sense, at least they may converse.

THOUGHTSTICKER allows users to maintain their own vocabularies. Any topic may have a series of names consisting of words or phrases, each of which is recognized to be associated with the particular topic. Recall that topics are represented by words and phrases; they are not the words or phrases themselves. Topics are the stable and agreed-upon (and therefore public) elements of concepts. Each user may assert a primary name to be used in the displays containing topics and entailments. Furthermore, THOUGHTSTICKER allows users to maintain a series of "contextures", each of which allows different names.

For example, as noted in Figure 7, the contextures called "PL" calls a particular topic **knowledge**, while the contexture "Holist" calls the same topic **knowables**. This particular use of the capability may seem pedantic, but the general capability is consistent for keeping individuals distinct, whether within one individual (within the contexture) or across individuals (among different contextures).

# 6. The Essence of Process: Micro Simulation of Lp

## 6.1 Knowledge Representation Display

Against the backdrop of the detailed description of THOUGHTSTICKER above comes the central issue of this thesis. This section presents the genesis of the thesis as a display "problem", the solution of which immediately and inexorably led to its extension into a micro confirmation of the existing macro theory of conversations.

## 6.2 Displays in THOUGHTSTICKER

The authoring process described in the previous chapter results in structures which reside inside of THOUGHTSTICKER. These are intricate networks of topics joined together by their entailments. It is possible and indeed useful for the author to represent these structures graphically.

### 6.2.1 Experimental Software Facility

An experimental facility was constructed in software to allow for a wide range of experiments. All of the power of windows and menu-driver user interfaces were brought to bear, resulting in a kind of laboratory in which many experiments could be performed and reproduced. The capabilities of this software is implied in Figure 8a through 8c, which contain the primary choice menus that were used to produce the results for the thesis.

Figure 8a shows the "top-level" menu; note especially how additional experiments are performed immediately, using the same parameter choices, by the "Next Experiment" menu selection. The remainder of the

window is covered ("tiled" in the modern parlance) with a series of snapshots of the dynamic display. This features was used to produce all of the output for the various Figures, by performing a screen dump to the laser graphics printer.

Figure 8b shows the result of choosing "Change which relations to display" from the previous Figure. This shows a set of available relations (whether from a test suite, or from an actual entailment mesh available from THOUGHTSTICKER); those in inverse video (black background) are those to be dynamically displayed. Variations may thus be tried in rapid succession.

Figure 8c shows the result of choosing "Major Overhaul" from the top-level menu in Figure 8a. This menu allows detailed modification of all simulation details, such as the nature of the force laws, relative strength of the forces, some display enhancements, *etc.*

The discussion below relies on reference to the successive figures as produced by the experimental software environment just described.

### 6.2.2 Discussion of the Programming

The Symbolics environment is an exemplary one in which to develop software of an experimental nature, where the results and implications are not known beforehand. Issues such as efficiency of calculation or size of database did not require any consideration for this thesis. Advantage was taken of the "object-oriented" programming features of the environment, to improve the software development cycle and make for efficient modifications and extension of features. Effort was made to provide a clear display with smooth refresh to give an exceptionally good feel for the dynamics of the interaction of the topic elements.

These display features were embedded into the Naive THOUGHT-STICKER interface for access by users, whether authors viewing the evolution of their structures, or learners seeing the structure of the subject matter during learning. A capability for hardcopy output, used in the creation of the Figures as direct screen printing to a laser graphics printer, was also incorporated.

Certain features of the simulation required careful consideration in the course of construction. Primarily, of course, the interpretation of Lp dynamics required caution in interpretation, to insure that CT was not being compromised or "fudged" to achieve some pre-ordained result. In fact, a number of schemes alternative to the one presented in detail above were tried, first to insure the robustness of the general approach by evaluating close alternatives, and second to confirm a proper mapping to Lp dynamics.

As a simple example, the generation of repulsive forces across the entire structure eliminates any possibility for ambiguity, and corresponds to a *post hoc* and global knowledge of the integrity (or not) of the structure. More subtle were alternative interpretations which did not preserve analogy as the basis of coherence; for example, favoring some topics in the relation above others or not providing a symmetric view of all topics within the coherence. These were not interpretations consonant with CT, they did not provide consistent results when applied over trials with various configurations, and neither did they exhibit the properties of CT as predicted at the macro level.

### 6.2.3 Coherence Displayed

Consider that topics in a entailment cohere, that is, they make sense together; they are in the same topological neighborhood. Concurrently,

these topics (if they are in a stable entailment which does not contradict with other entailments) are distinct; they are not blurred together or confused with one another. It is a great advantage to the user to display the relationships contained in the knowledge representation, as a means of understanding the existing structures as well as the implications of new ones.

Figure 9 simultaneously displays two coherent entailments which are distributive on the topic "ARI." The models for the two entailments inside of THOUGHTSTICKER are:

- "ARI is examining the use of CASTE for Training" and
- "ARI is an acronym for Army Research Institute."

### 6.2.4 Animated Interpretations of Topic Relations

THOUGHTSTICKER displays the structure of the knowledge representation as an animated sequence. Each topic is represented on the screen by its word or phrase, and lines connect topics contained in the same entailments. The topics are originally displayed at random positions, and thereafter they move smoothly around the screen. Figure 9a through Figure 9c show the result of such a dynamic interaction between the two entailments described above. The topics are animated according to the following rules:

- Topics in the same entailment are attracted to each other, and hence move toward one another over time; but also
- Topics in the same entailment are distinct, and so if they come in close proximity to other topics from the same entailment, they repel each other.

These two force processes are shown diagrammatically in Figure 10.

These two rules are parallels of the notions of coherence (attraction, same neighborhood) and distinction (repulsion, distinct entities). The addition of the dynamic element during simulation provides the third component of Lp, namely, process.

Figure 11 shows the effect on a larger set of relations.

### 6.2.5 Pruning Displayed

The interpretation of the Lp operation of Pruning is shown on Figure 12a through 12e, where the sequence gives some flavor of the dynamics. In addition to the above rules, an additional rule is imposed, which places a force on the topic "CASTE", drawing it up in entailment to the others. Again the topics are at first positioned randomly on the screen, the forces between each topic are computed and their positions are changed accordingly. Again, the dynamic simulation gradually becomes stable. The resulting hierarchy displays in graphical terms the dependencies that were inherent in the original network, or heterarchy.

### 6.2.6 Contradiction Displayed

Given just these rules, the question is asked whether these simple dynamics would display the behavior of, say, contradiction checking. Figure 13a shows an initial and random positioning of the topics of two entailments as modeled by:

- "ARI is examining the use of CASTE for Training" and
- "ARI uses PLATO for Training."

There is a contradiction contained in the entailments using the topics

["ARI" "CASTE" "Training"] and ["ARI" "PLATO" "Training"]. Figures 13b and 13c show intermediate, still pictures during the dynamic simulation. Finally, in Figure 13d, the resulting display shows that there is, in fact, no distinction between "CASTE" and "PLATO" as embodied in the entailments as they stand.

Note that the software has not examined the structure "globally", as it were, in the way that the contradiction checking does. Relationships are computed only within each entailment. The software simulation has merely imposed the simple rules described above, acting simultaneously on each topic in an analog to the meaning of the relationships as specified in Lp. The result is consistent with CT in that there is not enough distinction within the present entailments to maintain a distinction between the two topics, and so they occupy the same position on the screen. The addition of further distinction would eliminate the contradictory situation; for example, ["ARI" "CASTE" "Pask" "Training"] and ["ARI" "PLATO" "CDC" "Training"]. This example is shown in Figure 14a through 14c.

## 6.3 Conflict Terminology: Ambiguity and Contradiction

The two terms, ambiguity and contradiction, refer to the same cognitive situation; the term used is an observer's label.

"Ambiguity" emphasizes that there is a lack of available distinction between two (or more) topics; hence it is ambiguous which topic is indicated, or indeed whether there are two distinguishable topics instead of one. In the display of Figure 13d, the topics become ambiguous because they are indistinguishable from each other.

"Contradiction" emphasizes that when a production is begun with

particular topics, two (or more) entailments are activated and these processes conflict. It would be **contradictory** to produce two distinct topics from the same production of topics. Figure 13 can be interpreted as displaying the production from the same topics resulting in a conflicting or unknown result.

Since the observer names the cognitive event as ambiguity or contradiction, it is an error to be concerned with one or the other when conflict is detected by the Rule of Genoa as specified by Lp and embodied in, for example, THOUGHTSTICKER.

## 6.4 The Activation of Analogy versus Coherence

As noted in Section 5.5.3, analogy is a more fundamental form of Lp relation in that it is a pre-condition to the existence of coherence.

It is a basic issue to decide how to simulate an Lp structure. Clearly since analogy is more basic, it should be the basis for process activation. This is the case in the micro simulation presented, although it is clearer to describe the software in terms of coherences and hence the later sections take this perspective. In actuality, the simulation behaves like processes of analogy because it relates a given topic in the coherence to all of its neighbors at once, to compute its new relationship to them. This is an analogical relationship. It then proceeds to each of the other topics in the coherence in turn, representing in the end the complete set of analogies that must exist to form the stable structure and inter-relationship that is a coherence. It is because all of the necessary analogies exist that the complete coherence (a) produces relatively stable positions for the topics in the simulation and (b) shows the conflict points within those structures that contain them. These two points will be brought out in the detail below.

## 6.5 "Forces" Model

### 6.5.1 Movement toward Micro Modelling

The concept for computing Lp structures in the manner described above arose in two ways.

First, it arose from a desire to capture and compute the essence of Lp as a kinetics (Pask 1980d), restoring its status as a process model. This is a crucial distinction, of the kind which gives meaning to "simulate" versus "reify", and one which separates CT from AI. To continue research on Lp based on the static representations of THOUGHTSTICKER (which are adequate and practical for applications such as knowledge representation in training) would not lead to the substantial advantages that a kinetic model would; for example, there would be no potential for innovations arising within the computed structures themselves. It therefore seemed essential to me that this avenue be pursued.

Second, the process model for computing Lp structures was attractive for its parallels with the software models of "actor" semantics (Hewitt 1972; Hewitt & Baker 1977) as well as physics (Deutsch 1985). The "actor" models became popular with the increased research activity in parallel computing especially when the limitations of single-processor, "von Neumann" architectures became more apparent to workers in the field. These appealed to me, both because of interest in simulation-based computation, whether for graphics and animation, or for the extension of conventional models of computation. Most recently, the development of quantum-mechanical models (Deutsch 1985) has emphasized the need for alternative models of computation.

Here is provided an interpretation of stable entities of Lp as individual "actors" (in the sense that they are individual and separable) influencing

each other according to the relational organization between them. Thus there are "forces" acting between the entities, or topics, which influence their "motions" around each other. (Alternatively one may take the Einsteinian view that the "shape of space" is determined by the relations.)

Another characterization of the search space represented by the forces model is that of a minimum energy state which is sought by the interaction of the elements of the organization. The minimum energy state represents a configuration of the relations in the organization which is maximally stable, in that minimum energy is required to maintain it. Perturbations to the energy of the system in that state, without changes in organization, result in a convergent process back to the minimum energy state.

These acting forces determine the actors' kinetics, namely, their behaviors relative to each other as determined by the relations among them. As these actors represent topics and the organization represents cognitive relations, their resulting behavior is interpretable as a cognitive "result" as determined by Lp. The execution of the processes within this interpretation results in confirmation of the macro prediction within CT of such phenomena as pruning (see Section 6.5.4), conflict, conflict detection and resolution.

In the translation of the forces model into software there must come quantification of precisely how the entities are to interact, at what relative rates, *etc.* Unlike Newtonian mechanics where experiments may be performed to show the rates of gravitational acceleration, or in high-energy physics where the relative mass of tiny particles can be derived, Lp does not thus far provide a quantification of the forces involved. ("Arc-distance" is a measure of conceptual distance within a structure, and is reflected in the display results of the simulation; however this is a different quantity from what is being discussed here.) It is not

inconceivable that experiments could be done to derive some of these (see Section 7.3.3 for speculation along these lines). A series of experimental runs were performed to explore a range of possible interpretations of "forces", and these are given below.

## 6.5.2 Basic Force Calculations

The equations for these calculations are contained in Appendix A.

The equations used require position information in 2 dimensions, as usual called x and y. These positions are updated repeatedly, as fast as the simulation can run. There is a "time slice" parameter which determines the amount of time interval that is considered to have passed between the previous iteration and the current one, and this "delta-time" determines the overall rate of the simulation. There is no need for tracking the actual elapsed time between iterations because there is no need for mapping the simulation to clock time or any other "real time" considerations in the simulation.

It is interesting to consider that the value for delta-time is the basic "thermodynamics" of the system, a background energy relative to which all interaction takes place. Lower values require longer to come to stable configurations; higher values come to stability sooner but only after passing through more-highly energetic, and hence less stable, states.

The x and y positions are updated from velocity values, also computed in x and y. These in turn are modified each iteration by acceleration values for x and y. It is the acceleration values that are actually modified by the interactions of the entities in the simulation.

There are two forces at work in the simulation. The attractive force operates on those entities (topics) that exist in the same coherence. It

operates without much effect at a distance and with increasing effect as the distances between them decrease. Thus the attractive effect that they have is proportional to the distance between them. Each topic determines its distance to each other topic in the same coherence, and is accelerated toward each such topic in proportion to its distance.

The repulsive force also operates in proportion to distance, with closer distances creating greater repulsion, and again the result is applied for each relevant topic to the acceleration.

In both cases the calculations are performed in both x and y dimensions. After all such interactions are computed, the velocity and then the position of the topic are updated. The new position is used to plot the topic on the screen.

The precise effect that distance has is determined by an exponential parameter. For a value of 2, the simulation is a standard Newtonian inverse square law. For a value of 1, the simulation is a linear law. Both these values and some intermediates were used in a series of trials, as described in the next section.

Other parameters control further relative interactions, such as the relative strengths of the 2 major forces' interaction. After some experimentation it was seen that these could be kept equal and hence their absolute value is irrelevant since they are normalized throughout the equations.

A parameter was used to insure that long topic names would still appear left-to-right on the display and not interfere with other topic names, but this was used only for purposes of display clarity; all results were obtained without this additional calculation being performed.

It was found that a generalized "center-of-mass" offset was useful in insuring that the entire structure did not drift off screen. This computes

where the average of all topics on the complete display would be placed. The offset from this center of mass to the center of the screen is derived. Then, all of the objects are moved by that offset before display. Again, the primary results were computed without this adjustment, which was seen to be unnecessary in most cases anyway and was added later as a cosmetic enhancement when the micro simulation was added to THOUGHTSTICKER to display the knowledge structure as a user aid.

### 6.5.3 Linear and Squares Result

Coherent structures of varying complexity are easily displayed. Cases of contradiction were demonstrated for a variety of values for the exponential parameter, with a range of values between 1.0 and 2.0, representing the linear and square law result respectively.

It was seen that the end result was not significantly different for any value in this range; time to settle and slight overall variation in final distances were the only tangible differences. For the value of 1.0, the structure might spread outside the scope of the display screen; this could be prevented by the adjustment factor of the relative strengths of the 2 forces. However, the configurations were consistent with other values for the exponential greater than 1.0.

For values above 1.0 to 2.0, the end configurations were consistent with the exception of the absolute distances which resulted once the structure stabilized. This of course reflects the variation of balance of the forces. Beyond this difference, which does not effect the final result, only the time to stabilize and the range of motion of the topics during the stabilization time varied. As might be expected, the higher values for the exponential parameter led to higher energetics and longer stabilization times.

The following table presents a series of trials with significant cases of contradiction as detected by the Rule of Genoa. The results from the micro simulation are given with explanation.

Adicity refers to the number of topics in a coherence; hence 3-adicity indicates 3 topics, *etc*. The equal sign, "=", does not indicate equivalence but rather the lack of distinction between the topics related by the sign; for example, q = r indicates lack of distinction between the topics q and r. The phase "close to" means that although the topics do not fully overlap, they are quite close to each other and closer than any other pair in the structure.

Figure 15 in its various roman-numbered sub-figures contains each case in order.

Table 1: Contradiction Cases Results

| Case as determined in macro CT theory from Rule of Genoa: | Results computed from micro simulation of Lp: |
|---|---|
| I. Full Genoa (1 non-overlap) 3-adicity in 2 coherences: (t p d) and (t p e) | Full overlap of ambiguous topics detected in all trials: d = e |
| II. Full Genoa (1 non-overlap) 4-adicity in 2 coherences: (t p q d) and (t p q e) | Full overlap of ambiguous topics detected in all trials: d = e |
| III. Full Genoa (1 non-overlap) 5-adicity in 2 coherences: (t p q r d) and (t p q r e) | Full overlap of ambiguous topics detected in all trials: d = e |
| IV. Subset 4 adicity and 3 adicity: (t p r q) and (t p r) | No overlap |
| V. Partial Genoa 4-adicity and 3-adicity: (t p q d) and (t p r) | Partial overlap in 2 results: r close to d (shown) or r close to q or no overlap |
| VI. Partial Genoa 5-adicity and 4-adicity: (t p r e f) and (t p q d) | Partial overlap in 6 results: q close to f and d close to r (shown) or d close to f and q close to r or d close to f and q close to e or d close to e and q close to r or d close to e and q close to e or d close to r and q close to f |
| VII. Partial Genoa 4-adicity in 2 coherences: (t p q d) and (t p r e) | Full Overlap in 2 results: d = e and q = r (shown) or d = r and q = e |

### 6.5.4 Prune Case

The Lp operation of pruning has been achieved by the addition of a further force representing the hierarchical relationship between a head node (*i.e.* the focus of the pruning operation) and the remainder of the structure. This force acts as a further acceleration on the head node(s) only, in the vertical direction relative to the orientation of the display. The head node(s) drift upwards, attracting the topics to which they are related by coherence, pulling up others connected to those, and so forth. Because of the "center-of-mass" correction described above, the result did not drift up but rather arranged itself relative to the vertical axis.

An alternative interpretation of the Pruning operation might have been the sequential "activation" of each topic in sweeping arc distances down the structure. This could have been simulated in display by changing the representation of each topic, for example by moving from bold to non-bold characters, or tracking down the line connections between topics. However, each of these was seen to be computationally expensive and none would serve to clarify the display for the user. The interpretation of Pruning as an additional force is valid and consistent with the interpretation of the Lp relations of coherence and distinction as forces of attraction and repulsion.

The result is a display of pruning much like those constructed by hand from an entailment mesh. Of course topics may overlay each other if many coherences are processed at once. In this case an additional (but artificial) calculation may be made, forcing all nodes to be distinct. The result is a pleasing display for the user. One might attempt to justify the use of such an additional calculation by asserting that in a coherent mesh all topics are in fact distinct from one another; however this would compromise the very essence of the microscopic simulation in which

such a result comes from individual computations distributed locally throughout the structure. As emphasized earlier in Section 4.5, such a statement as to the distinction across the entire mesh is a global and macroscopic observation that can be made only from outside the structure.

An additional use of the pruning calculation is for the purpose of asserting a focus of attention for the user during the computation of coherence and distinction. For example, choosing the 2 topics around which an ambiguity exists in a case of Genoa conflict causes these topics to rise higher in the display than their neighbors, thereby giving prominence to the important parts of the situation at hand. Of course, knowing which 2 topics have this condition is a similar type of "global" knowledge. Note that choosing any 2 topics in this particular case (and any set of topics in any other case) does not alter the detection of ambiguity by the microscopic computation; rather, it affords an alternative focus of attention in the figure. All trials presented below were confirmed results for both head node/prunings and no such additional pruning force.

## 6.6 Discussion of Results

Figure 15 shows the detailed results of each of the following cases, numbered accordingly; for example, Case I, "Basic Contradiction Detection: Full Genoa" is Figure 15 I.

### 6.6.1 Basic Contradiction Detection: Full Genoa

Full Genoa is the case where only 1 topic in each coherent relation does not overlap with the other relation. It is the simplest case of contradiction.

As shown in Cases I, II and III, the micro simulation succeeded in

displaying contradiction in all cases of equal adicity of relation; results are given above up to adicity 5 but consideration of higher adicity and trials up to adicity 7 confirm this. This is as expected when considering the equality of forces in each relation and the symmetry of the computation across the relations.

### 6.6.2 Subset

The subset in Case IV might be surprising in that it showed no discernible overlap or even close proximity. However the result is consistent and confirms one class of result predicted in Lp for this case (Pask 1978). The overlapping topics themselves, although they are present in both relations indicated by the same name in the diagram, they in fact **cannot** be the same topic because they contribute to the relations of different adicities; therefore they must be **in part** different topics (the issue has been discussed most extensively in Clark 1980). Although the issue is not fully resolved so far as the theory is concerned, this micro simulation result would support the point of view that topics take some of their characteristics from the relation (*i.e.* in this case the relations adicity) they are present in.

### 6.6.3 Ambiguous contradiction: Partial Genoa

Partial Genoa exists in the macro theory whenever the extent of the ambiguity cannot be completely characterized; this has been called "ambiguously ambiguous" to distinguish from Full Genoa which is "unambiguously ambiguous" (Gregory 1982).

Case V correctly showed the closeness of the topics r and q, or r and d in the structure. However there was a third result in which no overlap occurred. This showed a limitation of the projection scheme used to

display the n-dimensional structure in the limiting 2-dimensions of the display. The resulting structure, depending on initial conditions, was a condition where the topics would find states in which the overlap would not occur due to forces keeping the ambiguous topics widely separated. This is due to the limitation of the simulation model in which the mapping to 2 dimensions occurs in the computation (in contrast to a computation in n dimensions that is then projected under user control; see Section 7.3.1). Topics as they settle in position interfere with others from "getting around" their neighbors to the true minimum configuration.

The probability of this occurring was lowered by increasing the delta-time parameter, representing the absolute thermodynamic energy of the system. This increased the energetics of the individual topics and encouraged motion away from the states of local minima.

Although this condition was present in some configurations, the basis of the entire simulation approach is not compromised, because within the dimensionality that is completely handled by the present calculations, all results were consistent with CT. Only cases beyond the dimensionality of the present software resulted in some trials in a partial result. The full, n-dimensional computation would avoid this problem and allow for much more complex computation than could be shown by the present forces model.

Case VI did not contain such non-minimal cases and provided consistent results.

Case VII, where the ambiguity is increasing, provided a consistent result in which the mapping of which topics were ambiguous with which, was shown.

# 7. Conclusions & Summary

## 7.1 Lp Software at the Macro Level

THOUGHTSTICKER is a software manifestation of the calculus Lp, itself the dual of Conversation Theory. It has been argued in this thesis that THOUGHTSTICKER exists at a "macro" level, relative to any true embodiment of Lp involving the process component in addition to the coherence and distinction components that THOUGHTSTICKER already possesses. Even at this macro level, it is a substantial enhancement to previously existing software systems. This derives from two classes of enhancement: those taken from Conversation Theory, and those invented or developed in the course of writing Conversation Theory into THOUGHTSTICKER code.

The strengths of THOUGHTSTICKER derived from CT come from the cognitive basis of CT. Modelling knowing [*sic*] is substantially improved above other AI techniques because of this. The resulting software provides advantages above "thought processors" and computer-aided instruction systems because of the stimulation provided the author during the knowledgebase creation process, including indication of existing topics, detection of potential conflict, and saturation. The resulting knowledgebase has properties that make it particularly suitable for exploration in a manner consistent with a variety of conceptual styles. These advantages to THOUGHTSTICKER derive from its origins.

THOUGHTSTICKER strengths derived from its implementation are many and various. Some have to do with the advantages of modern menu-driven MMI, with a mouse pointing device, multiple windows and panes on the same screen, bit-mapped, high-resolution displays and so forth. The advantages too of object-oriented programming, for rapid

prototyping and swift experimental change of features has also aided the implementation substantially. Conceptual features, however, make up the bulk of advantages of THOUGHTSTICKER. Unique to this implementation, these are:

- A full set of tools for the detailed manipulation of the Lp structures.

- An evolving set of high-level, semi-automatic procedures for converting conventional courses to THOUGHTSTICKER structures, examining the implications of various tutorial strategies on any course, searching for lack of uniformities in the structures, *etc.*

- Complex heuristics for providing a many-dimensional classification and delivery of training based on conceptual styles.

- A true personalization of the user interaction based on a complete history of interaction between the user and the system.

- Facilities to manage the problems of multiple authors.

These advantages of THOUGHTSTICKER derive from the ingenuity of the implementation as constructed by Jeffrey Nicoll and myself. A more detailed breakdown of the responsibilities, for the purpose of documentation and with the understanding that all such histories can only be coarse in nature, is contained in Appendix C.

## 7.2 Macro theory and Micro confirmation

A microscopic simulation of the forces within a relational structure representing the relationship of entities within Lp is seen to exhibit certain behaviors. These behaviors were already under consideration in the macro theory of CT. There, ambiguity and conflict were detected by the calculation of certain structural relationships, a calculation performed from "outside" the system by a process (perspective, individual, observer) that had access to the entire structure. The microscopic simulation, which does not perform globally and has only local information, results in configurations interpretable as ambiguity and/or conflict in the same cases as indicated in the macroscopic theory.

Therefore, the macro behaviors which had previously had the status of observable events in the cognitive domain as described within CT now can be computed directly from the microscopic processes dictated by the protologic Lp. Although Lp was drawn from experience of CT, before these experiments it had no independent and empirical confirmation. In addition, evidence is provided for resolving the open question within CT on the interpretation of "sameness" of topics as dependent on the adicity of their relations.

Conflict is the name given to a condition observed from outside the system. It arises perforce in situations of contradiction/ambiguity where the existing structure is unstable and where some changes to the structure result in stability while others do not.

The locus within the structure where such changes must be made can be performed by a macro process from outside the system or a micro process from inside. Theories, of which CT is one, can provide the means for the macro calculation and there is little magic in this: all of the information is known globally. Such theories are useful for *post hoc* explanations of

how a system behaved; they are however useless in creating such change within a system.

Systems contain distinctions in two senses. Primarily, they contain the distinctions attributed by observers (Pask 1963). However, systems that innovate must be capable of creating distinctions from within; otherwise nothing new can arise. Therefore any cognitive theory must provide for a mechanism whereby distinctions arise within the system rather than outside as imposed by an omnipotent observer. In other words, the system must, within its own structure, be capable of computing sufficient similarities and distinctions to create a separable observer; this is tantamount to saying that it is capable of computing the new distinction.

It is therefore crucial for any theoretical framework to show how distinctions can arise, as well as explain them once they have. Conversation Theory and its dual, Lp, is one such framework.

## 7.3 Extensions to the Work

### 7.3.1 Dimensional Control

For cases involving more than a few coherences, the display of the result as a projection onto 2 dimensions can be cluttered and the bifurcation point can be obscured. Because the dimension of the structure is greater than 2 dimensions, the problem cannot be avoided without the ability to project onto more than 2 dimensions, impractical for the present technology and even if soluble for low dimensions is certainly not practical to display for higher dimensions. An alternative would be to perform the calculation in dimensions that are relative to each structural relation (rather than simply in x and y) and to then control which dimensions are projected onto the 2 dimensions of the display. This control could be determined by the user, which may be useful in scanning

for interesting structures and points of potential future bifurcation. It would also be possible and desirable to provide an automatic software control, determining the projection dynamically as a result of the condition of the simulation. The condition(s) of interest would be in part controlled by the user. In some conditions, the user's interest will be in maximizing the distance between certain topics, to determine their entailment. In others, the intentional overlay of relations would allow for the extraction of similarities and differences as represented by the topics entailed in the relations. Such a "driving through knowledge" would be a powerful tool for the user, and later point to the means for automating certain operations (such as saturation) on the user's behalf.

### 7.3.2 Cognitive Force Values

It is conceivable that the relative forces, simulated above for values from linear to square-law, could be more precisely quantified, and specific values determined, experimentally. The experimental result would need to be correlated across a number of individual trials and subjects but might provide a higher degree of veracity to the simulation. For example, the role of the number of topics in a relation (the adicity) on the rapidity of detection of conflict might allow the derivation of the exponential parameter. The relative balance of the adicity in the relations involved in conflict might have the effect of speeding or slowing the process as well.

The influence of analogies and generalizations that provide additional connections, and hence influence on the structure, might also be involved in quantifying the parameters. Such quantification would probably be necessary in the further computation of condense/expand by micro simulation.

### 7.3.3 Pruning and Resolution

The dimensional extensions discussed above would allow for greater information to be drawn from pruning cases. For example, suppose that multiple head nodes were chosen and were accelerated in opposing directions in a dimension in addition to those already allocated to the existing relations. Eventually in the computation, certain topics would be pulled in that dimension to the point where the strain could be detected by calculation. This could be indicated graphically to the user, who could choose to insure that topic remain a single entity, or could allow for bifurcation. Upon the allowance for bifurcation, the implications would then spread through the structure and further places for possible bifurcation would be indicated by the same computation. The result would be a highly efficient and visually exciting means for extending the structure through conflict detection and resolution.

### 7.3.4 New Hardware

The rise of new hardware architectures that allow for massive parallel computation provide the means for exploring the implications of the microscopic computational model offered in this thesis. It would become practical to perform THOUGHTSTICKER operations on massive entailment structures by microscopic simulation rather than the current, compromised macro calculations. This would allow for continuous management of the knowledge, especially as its creation and evolution becomes distributed throughout large and geographically disparate electronic networks.

By far the most exciting prospect however is the relaxation of the restriction to parallel computation to concurrent computation. This is becoming more feasible in the newest hardware architectures (Hillis

1985). Now the potential is to create a true Lp engine, capable of evolution and innovation within itself. The microsimulation proposed here, as extended to include condense/expand operations including generalization, would be the basis for such an engine.

# Appendices A through F

# Appendix A
# Equations

## Micro simulation of Lp coherence structure

Where:

$C_\alpha$ is a coherence consisting of topics a, b, ... i ... n ... , z

i is a given topic in $C_\alpha$ for which a New-position is to be calculated

n is another topic in coherence $C_\alpha$

$x_i$ is the Old-position (current position) in the x dimension for topic i

$y_i$ is the Old-position (current position) in the y dimension for topic i

$x_n$ is the Old-position in the x dimension of neighbor n

$y_n$ is the Old-position in the y dimension of neighbor n

Delta-Time is a time-slice parameter

Angle is the angle of topic n relative to topic i

Exponential-Parameter has values from 1.0 to 2.0

Attraction-Factor and Repulsion-Factor have values from 1.0 to 2000.0

Then:

For all $x_n$ for n = {a, b, c, ... z} for all neighbors of i in $C_\alpha$

Distance = SQRT $[(x_i - x_n)^2+(y_i - y_n)^2]$

Attraction $x_{i,n}$ = Distance $^{-\text{Exponential-Parameter}}$ * cos (Angle) * Attraction-Factor

Repulsion $x_{i,n}$ = Distance $^{+\text{Exponential-Parameter}}$ * cos (Angle) * Repulsion-Factor

And, for all $x_j$, where j = {a, b, c, ... z}

New-Acceleration $x_i$ = + Attraction $x_{i,j}$ - Repulsion $x_{i,j}$

New-Velocity $x_i$ = Old-Velocity $x_i$ + (Delta-Time * New-Acceleration $x_{i,j}$)

New-Position $x_i$ = Old-Position $x_i$ + (Delta-Time * New-Velocity $x_{i,j}$)

For all $y_n$ for n = {a, b, c, ... z} for all neighbors of i in $C_\alpha$

Attraction $y_{i,n}$ = Distance $^{-\text{Exponential-Parameter}}$ * sin (Angle) * Attraction-Factor

Repulsion $y_{i,n}$ = Distance $^{+\text{Exponential-Parameter}}$ * sin (Angle) * Repulsion-Factor

And, for all $y_j$, for j = {a, b, c, ... z}

New-Acceleration $y_i$ = + Attraction $y_{i,j}$ - Repulsion $y_{i,j}$

New-Velocity $y_i$ = Old-Velocity $y_i$ + (Delta-Time * New-Acceleration $y_{i,j}$)

New-Position $y_i$ = Old-Position $y_i$ + (Delta-Time * New-Velocity $y_{i,j}$)

# Appendix B
# Software Program Listings

The experiments for software constructed for the micro, dynamic Lp simulation of this dissertation was begun on a Commodore "PET" microcomputer, written in "PET" BASIC, in 1981 and 1982. Simulations were performed dynamically and displayed as time slices in a graphical arrangement on the 52-column display. Hardcopy was available, along with a variety of timing and running modes. Linear and Square-law calculations were prototyped and experiments run.

The simulation was re-coded on an early vintage LISP Machine introduced by Symbolics, Inc. in 1982, the Model 3600, Serial #129 (*i.e.*, number 29, as they start at 100). The source code program listing following the basic calculations of the PET version was written in ZetaLISP (technically, "Old-ZetaLISP") in 1983. It uses the graphics handling of that environment and the Flavors system of object oriented programming. The code was written during the very early days of experience with the machine and hence is not an exemplary use of the modern LISP dialects.

```
(defun test()
  (*catch 'out
    (tv:menu-choose
      '(("the only choice" eval (tv:choose-variable-values
                                  '((*c1 "this one" :assoc ,*vary-values)
                                    (*c2 "next one" :assoc (("first" . ,*c1) ("second" . 2))))
                                 :margin-choices '("thats all" ("abort" (*throw 'out nil)))
                                 ))
        ))))


(Defun display-kr-mod()                          ; menu driven changes
  (tv:menu-choose
    '(("" :no-select 'ignore) ("" :no-select 'ignore)
      ("  Next Experiment  " :value nil :documentation "A further trial with current conditions.")
      ("" :no-select 'ignore)
      ("  Select mesh  " :eval (pick-mesh-subset))
      ("" :no-select 'ignore)
      ("  Change which relations to display  " :eval (pick-kr-coherences) :documentation "Under Debugging")
      ("" :no-select 'ignore)
      ("  Change Conditions  " :eval (change-kr-conditions) :documentation "Change the conditions under which the Experiment is performed.")
      ("" :no-select 'ignore)
      ("   Major Overhaul   " :eval (old-change-list))
      ("" :no-select 'ignore)
      ("  Exit  " :eval (setq *exit-flag T) :documentation "Exit back to whence you came.")
      ))
  (setq *wipe-cycle *wipe-factor)
  )

(Defun pick-mesh-subset()
  (setq *kr-coherence-choices* (if (setq *real-kr-mesh* (get-meshes)) (get-coherences-from-mesh *real-kr-mesh*)
                                 *default-coherence-choices*))
  (unless *kr-coherence-choices* (setq *kr-coherence-choices* *default-coherence-choices*))     ; safety fudge
  (setq *kr-coherences-chosen* nil)              ;maybe this is why menu is crashing
  (pick-kr-coherences)
  )

(Defun pick-kr-coherences()
  (if (null *kr-coherence-choices*) (setq *kr-coherence-choices* (get-coherences-from-mesh *real-kr-mesh*)))
  (setq *kr-coherences-chosen* (kr-coherence-chooser *kr-coherence-choices*  *kr-coherences-chosen*
                                                     "  Subset of Mesh to Display  ") )
  (when *hierarchy (pick-head-topics)))

(defun kr-coherence-chooser (all-coherences coherences-already-chosen label &aux picks)
  "Menu for coherences which when chosen are returned."
  (setq picks (tv:make-window 'tv:momentary-multiple-menu :font-map '(fonts:medfnb fonts:hl12i)
                              :item-list (loop for each-coh-list in all-coherences
                                               collect
```

```
                                        ;; take ("a" "b" "c") and make "a b c"
                                        (list (loop for str in each-coh-list with agg = " [ " do
                                                    (setq agg (string-append agg " "  str " " (string 13.)))
                                                    finally (return (string-append (string-right-trim
                                                                                     (string-append (string 13.) " ")
                                                                                     agg)
                                                                                   " ] ")))
                                              :value
                                              each-coh-list))
                            :label label))
  (send picks :set-highlighted-values coherences-already-chosen)
  (loop for each in (send picks :choose) until (null each) collect each)        ;stupid way to drop trailing nils
  )



(defun pick-head-topics(&aux topics picks sub-l)
  (setq topics (sort (remove-dup-strings (loop for each in *kr-coherences-chosen* appending each)) 'string-lessp))
  (setq picks (tv:make-window 'pan-momentary-menu
                              :item-list topics
    :label "  Which topics to be drawn up by the hair?  "))
  (if (setq sub-l (sub-lister *string-topics* topics)) (send picks :set-highlighted-values sub-l))
  (setq *string-topics* (send picks :choose))
  )

(defun sub-lister (list common-list)    ;give those in common
  (loop for each in list when  (member each common-list) collect each))

(defun remove-dup-strings (list &aux (l nil))
  (loop for each in list unless (member each l) do (push each l))
  l)

(Defun change-kr-conditions()
  (*catch 'kr-conditions-out
    (tv:menu-choose '(,*dead-mouse*
                      ,*dead-mouse*
                      ("   Major Overhaul   " :eval (old-change-list))
                      ,*dead-mouse*
                      ("  Pick Head topics" :eval (progn (setq *hair-list-hierarchy* T)
                                                         (pick-head-topics)))
                      ("  (experimental) Hierarchy Yes (pruning)    " :eval (progn (setq *hair-list-hierarchy* T)))
                      ("  Heterarchy Yes  " :eval (setq *hair-list-hierarchy* nil))
                      ("  Orient Dimension  " :eval (shift-neighborhood-dimension))
                      ,*dead-mouse*
                      ))
    ))

(defun old-change-list()
  (tv:choose-variable-values
      '(""
        (*delta-time "  Rate" :number)
        ""                                          ;(*velocity-decay "  Type of interaction" :assoc (("   True Forces" . .9) ("   Linear Displace
ment" . 0.0)))
        (*distinct-in-mesh "  Compute Distinction across entire mesh" :boolean)
        (*distinct-in-neighborhood "  Compute Distinction across coherences" :boolean)
```

```
        (*distinct-r "  R-law to use (1=linear 2=usual Newtonian)" :number)
        ;; (*distinct-delta "  Distinction distance in characters" :number)
        (*distinct-factor "  Distinction strength" :number)
        " "
        (*attract "  Compute Attraction within each coherence" :boolean)
        (*attract-r "  R-law to use (1=linear 2=usual Newtonian)" :number)
        ;; (*attract-delta "  Attraction distance in characters" :number)
        (*attract-factor "  Attraction strength" :number)
        " "
;       (*hierarchy "  Display Hierarchy" :boolean)
        (*head-trails* "  Make trails of head node of Pruning" :boolean)
;       ""
        (*hair-list-hierarchy* "  Compute only the chosen head nodes" :boolean)
        (*hair-pull-factor* "  How strongly" :number)
        ""
        (*each-coherence-hierarchy* "  Compute Hierarchy within each coherence" :boolean)
        (*hierarchy-r "  R-law to use (1=linear 2=usual Newtonian)" :number)
        ;; (*hierarchy-delta "  Hierarchy distance in characters" :number)
        (*hierarchy-factor "  Hierarchy strength" :number)
        " "
        " * * * * * * * * * * * * * * *"
        (*position-random "  How random an initial position (higher value for more spread)" :number)
        ;; (*wipe-factor "  How many frames to keep before clear" :number)
        (*velocity-still "  How still before considered stopped (nil for never)" :number-or-nil)
        (*max-velocity-decay "  Viscosity of ether (lower values are slower)" :number)
        (*name-length-factor "  Compensation for long topic names (higher yields more spread)" :number))
      :label "                  Node Orbit Experiments"
      :margin-choices '("New Experiment" ("Continue" (progn (setq *continue-flag T) (*throw 'kr-conditions-out T))))
      ))

(Defun kr-display ()
  (begin)
  (setq *start-x (// (send *krwindow :width) 2))
  (setq *start-y (// (send *krwindow :height))) ;these should be sends to window for width
  (setq *next-trial T)
  (setq *exit-flag nil)
  (prog()
    a (if *next-trial (go b))
      (do-doc "Node-Orbit Experiment for Entailment Mesh Display" *krwindow)
      (setq *continue-flag nil)
      (display-kr-mod)
      (if *continue-flag (go d))
      (if *exit-flag (go e))
    b ;; (setq *but-pushed 0)
      (setq *next-trial nil)
      (begin)
    d
      (Node-action)
      ;; (send *krwindow :home-down)
      (go a)
    e ;; (print "Type (kr-display) to LISP to restart.")
      )
  )

(Defun node-action(&aux but-pushed)
  (setq *node-move T)                          ;insure the loop once
```

```
  (*catch 'node-action-out
    (do-doc "Node-Orbit Experiment:   Mouse Hold Left for another trial.    Hold Middle to grab node.   Hold Right for menu."
            *krwindow)
    (do ()
        ((or (not *node-move) (exit-buttons *krwindow)))
;;      (send *krwindow :clear-window)
      (dotimes (i *wipe-cycle)
        (send *kr-mesh :center)
        (send *kr-mesh :orbit)
        (dolist (neighborhood (send *kr-mesh :neighborhoods))
          (when (equal (setq but-pushed (get-buttons)) 4) (setq *next-trial nil) (*throw 'node-action-out t))   ;changes
          (when (equal but-pushed 2) (node-grabber) (setq *old-buttons 0))
          (when (equal but-pushed 1) (setq *next-trial t) (*throw 'node-action-out t))
          (send neighborhood :orbit))
        (setq *node-move nil)
        (send *kr-mesh :erase)
        (send *kr-mesh :lp-it)
        (send *kr-mesh :show)
        )))  )

(Defun node-grabber()
 ;;  (try-init)
  (loop until (equal (get-buttons) 0) do
        (loop for node in (send *kr-mesh :nodelist)
              do (send node :grabbed (- tv:mouse-x *x-offset) (- tv:mouse-y *y-offset)))
        )
  )

(Defun begin (&optional (string-coherences *kr-coherences-chosen*))
  (if (null *krwindow)
      (kr-window)
      (send *krwindow :expose))
  (multiple-value-bind (l to r b) (send *krwindow :inside-edges)
    (setq *char-width (send *krwindow :character-width (character "x")))
    (setq *char-height (send *krwindow :line-height))
    (setq *x-left (+ (setq *x-offset l) (* *char-width 10)))
    (setq *x-right (- r (* *char-width 10)))
    (setq *y-top (+ (setq *y-offset to) (* *char-height 1)))
    (setq *y-bottom (- b (* *char-height 4))))
  (send *krwindow :clear-window)
  (setq *counter 0
        *solar-system nil
        *nodes nil
        *neighborhoods nil
        *kr-mesh (make-instance 'kr-mesh))
  ;; make the neighborhoods from chosen cohrences
  (process-coherences string-coherences)
  (process-head-topics)
  )

(defun process-coherences (string-coherences)
  "Converts coherences in string form to nodes etc. *Solar-system, etc, is not cleared so existing nodes are not reproduced."
  (loop for each-coh in string-coherences do (make-string-neighborhood each-coh)))

(defun process-head-topics ()
  "Sets *kr-head-topics* to the nodes, based on *string-topics*."
```

```
  (setq *hierarchy (or *each-coherence-hierarchy* *hair-list-hierarchy*))       ;set common flag for display
  (if (and *hair-list-hierarchy* (null *string-topics*)) (pick-head-topics))    ;be certain it has heads
  (setq *kr-head-topics* (setq *save (loop with string-topics-list = *string-topics*
                                           for topic-instance in (send *kr-mesh :nodelist)
                                           when (member (send topic-instance :text) string-topics-list) ;these are "a" "q" ...
                                           collect topic-instance))))


(Defun make-string-neighborhood (list-of-strings &aux i t1 t2
                                 (this-list-of-node-instances nil) tem xc yc)
  (multiple-value (xc yc) (screen-center *krwindow))
  (dolist (s list-of-strings)
    (multiple-value (i t1 t2 )(intern s))
    (if (setq tem (assoc i *solar-system))      ;('name-of-node #<NODE 7878>) pair
        nil                                    ;is in *solar-system, so dont make new one
        (setq tem (list i (make-instance 'node
                                         :xp xc :yp yc
                                         :text s)))     ;create new instance
        (push (second tem) *nodes)
        (send *kr-mesh :add-node (second tem))
        (push tem *solar-system))
    (push (second tem) this-list-of-node-instances))
  (setq this-list-of-node-instances (reverse this-list-of-node-instances))      ;make head the first
;;  (print "head node:")(print (send (first this-list-of-node-instances) :text))
  (push (make-instance 'neighborhood :nodelist this-list-of-node-instances) *neighborhoods)
  (send *kr-mesh :add-neighborhood (first *neighborhoods)))

(defun accumulate-names(list &aux (final-string ""))
  (loop for each in list do (setq final-string (string-append final-string "  " each)))
  final-string)

#|(Defun make-neighborhood (nodenames &aux (this-list-of-node-instances nil) tem )      ;obsolete
  (dolist (i nodenames)
    (if (setq tem (assoc i *solar-system))      ;('name-of-node #<NODE 7878>) pair
        nil                                    ;is in *solar-system, so dont make new one
        (setq tem (list i (make-instance 'node
                                         :xp *start-x :yp *start-y
                                         :text (get-pname i)))) ;create new instance
        (push (second tem) *nodes)
        (send *kr-mesh :add-node (second tem))
        (push tem *solar-system))

    (push (second tem) this-list-of-node-instances))    ;say you've got it in this neighborhood anyway

  (setq this-list-of-node-instances (reverse this-list-of-node-instances))      ;make head the first
;;  (print "head node:")(print (send (first this-list-of-node-instances) :text))
  (push (make-instance 'neighborhood :nodelist this-list-of-node-instances) *neighborhoods)
  (send *kr-mesh :add-neighborhood (first *neighborhoods)))|#

(defun shift-neighborhood-dimension (&aux neighborhood)
  "Allos choice of neighborhood and chang of dimension mapping of that one."
  ;; indicate which neighborhood
  (setq neighborhood (kr-coherence-chooser *kr-coherences-chosen* nil " Which neightborhood to shift: "))

  )
```

```
(defFlavor spun-mixin ((theta 0.0)
                       (spun-radius *initial-spun-radius)
                       (spun-rate *initial-spun-rate))
          ()
  :settable-instance-variables)

(Defflavor node (xp yp (xv 0.0)  (yv 0.0)  (xa 0.0)  (ya 0.0) text x-char-offset ;this in pixels
                 id                          ; unique, set from *counter and used to position them sometimes
                 neighborhood
                 old-xp old-yp)
          (spun-mixin)
  :settable-instance-variables)

(DefMethod (node :init)
          (&rest ignore)
  "Adjusts position params according to length of name, does initial display, sets old-positions for x, y, sets id."
  (setq id (setq *counter (+ 1 *counter)))
  (setq x-char-offset (* -1.0                ;negative
                         (// (string-length text) 2.0) *char-width))
  (setq old-xp xp) (setq old-yp yp)          ;what does this do?
  )

(defMethod (node :after :init)
          (&rest ignore &aux (head-offset (* *char-width 3.)) (personal-offset (* *char-width id)))
  "Repositioning: if part of *string-topics*, put in center; all others below."
  (cond (*init-random-position
         (setq xp (+ (startup) xp)) (setq yp (+ (startup) yp)))
        (*string-topics*
         (loop for node-name in *string-topics*
               do (setq xp (+ xp personal-offset))
               when (equal node-name text) do (setq yp (- yp head-offset))
               ;; else do (setq yp (+ yp personal-offset))
               ) )
        (T (tv:pop-up-message "no head node yet declared; this message in node :after :init") ))
;;  (send self :center)
  (send self :show)
  )

(defMethod (spun-mixin :after :init)(&rest ignore)
  "Sets x,y inits better."
  (when *spun-enabled (setf (send self :xp) (+ (send self :xp) spun-radius))))

(DefMethod (node :show)          ; very icky on font stufff
                                 ; get longer call to :draw-string for font set from instance variable
          ()
;;  (send *krwindow :set-cursorpos (fix (+ xp x-char-offset)) (fix yp))
  (cond-every ((and *head-bold* (member text *string-topics*))
               (send *krwindow :set-current-font 1))    ;*bold-font*
              (T
                (send *krwindow :draw-string text (fix (+ xp x-char-offset)) (fix yp)))
              ((and *head-bold* (member text *string-topics*))
               (send *krwindow :set-current-font 0)))   ;*non-bold-font*
;  (send *krwindow :draw-circle (fix xp) (fix yp) 2)
  )

(Defmethod (node :erase)                 ; the elegant one?
```

```
          ()
;;  (send *krwindow :set-cursorpos (fix (+ old-xp x-char-offset)) (fix old-yp))
  (cond-every ((and *head-bold* (member text *string-topics*))
               (send *krwindow :set-current-font 1))    ; *bold-font*
              ((or (null *head-trails*) (not (member text *string-topics*)))
               (send *krwindow :draw-string text (fix (+ xp x-char-offset)) (fix yp))
                                    ; (send *krwindow :string-out "        ")
               )
              ((and *head-bold* (member text *string-topics*))
               (send *krwindow :set-current-font 0))    ; *non-bold-font*
              ))

(Defmethod (node :add-xa)
           (increment)
  (setq xa (+ xa increment)))

(Defmethod (node :add-ya)
           (increment)
  (setq ya (+ ya increment)))

(defMethod (node :center)()
  (when *centering-enabled
    (setq xp (- xp (send *kr-mesh :offset-center-x)))
    (setq yp (- yp (send *kr-mesh :offset-center-y)))))

(Defmethod (node :newton-it)                    ;displacement version
           ()
  (setq old-xp xp) (setq old-yp yp)             ;save for erasing
  (setq xv (+ xv (max-mag xa *max-acceleration)))
  (setq xp (+ xp xv))
  ;;(setq xp (max (min xp *x-right) *x-left))

  (setq yv (+ yv (max-mag ya *max-acceleration)))
  (setq yp (+ yp yv))
  ;;(setq yp (max (min yp *y-bottom) *y-top))   ;!!! these were order-switched to try displacement

  (setq xv (* xv *velocity-decay))
  (setq yv (* yv *velocity-decay))
  (if (numberp *velocity-still)                 ;if nil then dont stop; for displacement fudge *velocity-still=0
      (when (or (or (> (abs xv) *velocity-still) (> (abs yv) *velocity-still))
                (or (> (abs xa) *acceleration-still) (> (abs ya) *acceleration-still))
                )
        ;;(> (distance xv yv 0 0) *velocity-still)
        (setq *node-move T)
        )
      (setq *node-move T))                      ;dont stop on velocity still ever
  (send self :center)
  (setq xa 0.0) (setq ya 0.0)
  )

(defun max-mag(x  max-value)
  "If abs of x > max-value then return max-value with proper sign; else return x."
       (* (if (minusp x) -1. +1.) (abs (min (abs x) (abs max-value)))))

(Defmethod (node :grabbed)
           (mx my)
```

```
  (when (inside mx my
                (- xp (max 20 (abs x-char-offset))) (- yp (* 3 *char-height))
                (+ xp (max 20 (abs x-char-offset))) (+ yp (* 3 *char-height)))
    (send *kr-mesh :show-links)(send self :erase)
    (setq xp mx)
    (setq yp my)
    (send *kr-mesh :show-links)(send self :show)
    ))

;;; ------------------------------- MESHES
(Defflavor kr-mesh ( (nodelist nil) (neighborhoods nil) (offset-center-x 0) (offset-center-y 0) screen-center-x screen-center-y)
           ()
  :settable-instance-variables)

(defMethod (kr-mesh :after :init)(&rest ignore)
  "Sets screen centers."
  (multiple-value-bind (cx cy) (screen-center *krwindow)
    (setq screen-center-x cx
          screen-center-y cy)))

(Defmethod (kr-mesh :add-neighborhood)
        (newneighborhood)
        (push newneighborhood neighborhoods))

(Defmethod (kr-mesh :add-node)
        (newnode)
        (push newnode nodelist))

(DefMethod (kr-mesh :show-one-level-derivation)
           (&optional (alu tv:alu-xor))
  (dolist (neighborhood *neighborhoods)
    (send neighborhood :show-one-level-derivation alu))
  )

(Defmethod (kr-mesh :show-neighborhood)
           ()
  (dolist (neighborhood *neighborhoods)
    (send neighborhood :show-neighborhood))
  )

(Defmethod (kr-mesh :show)
           ()
  (send self :show-links)
#|  (dolist (i nodelist)
    (send i :show))|#
  )

(defMethod (kr-mesh :center)()
  "Centers all nodes."
  (when nodelist
    (loop for node in nodelist with l = (length nodelist)
          with ax and ay
          summing (send node :xp) into x
          summing (send node :yp) into y
          finally (setq ax (// x l) ay (// y l))        ; averages
                  (send self :set-offset-center-x (- ax screen-center-x))
```

```
                (send self :set-offset-center-y (- ay screen-center-y)))))


(Defmethod (kr-mesh :show-links)
           ()
  (if *hierarchy (send self :show-one-level-derivation tv:alu-xor)
      (send self :show-neighborhood)))

(Defmethod (kr-mesh :lp-it)
           ()
  "For each nodelist, does erase//newton-it//sho cycle."
  (if *hair-list-hierarchy* (hair-list-hierarchy))      ;right place for this?
  (dolist (l nodelist)
    (send l :erase)
    (send l :newton-it)
    (send l :show)
    )
  )

(Defmethod (kr-mesh :erase)
           ()
  (send self :show-links)
#|  (dolist (l nodelist)
    (send l :erase))|#
  )

(Defmethod (kr-mesh :cycle)
           ()
  "Neutral cycler that dos it all."
  (setq *node-move nil)
  (send *kr-mesh :orbit)
 ; (send *kr-mesh :center)
  (loop for n in neighborhoods do (send n :orbit))
  (send *kr-mesh :erase)
  (send *kr-mesh :lp-it)
  (send *kr-mesh :show))

(defMethod (kr-mesh :redisplay)
           ()
  "Shows the present state for a blank screen."
  (loop for n in nodelist do (send n :show))
  (send self :show))

(Defmethod (kr-mesh :orbit)
           (&aux xp1 xp2 yp1 yp2 f r theta tf)
  "Computes *distinct-in-mesh if this flag is set."
  (Cond-every (*distinct-in-mesh
               (do ((list nodelist (cdr list)))
                   ((< (length list) 2))
                 (do ((b1 (first list)) (b2 (cdr list) (cdr b2)))
                     ((null b2))                ;          (print (send b1 :text))(print (send (first b2) :text))
                   (setq xp1 (send b1 :xp)) (setq xp2 (send (first b2) :xp))
                   (setq yp1 (send b1 :yp)) (setq yp2 (send (first b2) :yp))

                   (if (> (setq r (distance xp1 yp1 xp2 yp2))
                          (* *distinct-delta *char-width))     ;
```

```
                        nil                     ;then, no collide
                        (setq f (// *distinct-factor
                                    (expt (if (< r 1.0) 1.0 r) *distinct-r)))   ;computer force= r|2 * factor for beyond
                        (setq theta (if (and (equal xp1 xp2) (equal yp1 yp2))
                                        0.0
                                        (atan (abs (- yp2 yp1)) (abs (- xp2 xp1)))))
                        (setq tf (* (expt (string-length (send b1 :text)) *name-l:      :tor) f *delta-time))
                        (send b1 :add-xa (* (cos theta) (if (minusp (- xp1 xp2)) -       :f))
                        (send b1 :add-ya (* (sin theta) (if (minusp (- yp1 yp2)) -1. +1.; tf))
                        (setq tf (* (expt (string-length (send (first b2) :text)) *name-length-factor) f *delta-time))
                        (send (first b2) :add-xa (* (cos theta) (if (minusp (- xp2 xp1)) -1. +1.) tf))
                        (send (first b2) :add-ya (* (sin theta) (if (minusp (- yp2 yp1)) -1. +1.) tf))
                        ))))))

(Defflavor neighborhood (nodelist               ; list of nodes in the neighborhood
                         (ip-object nil)        ; instance of the originator, if any
                         (xd 0.0) (yd 0.0) (zd 0.0))   ; shift of dimension
          ()
  :initable-instance-variables
  :settable-instance-variables
  :gettable-instance-variables)

(Defmethod (neighborhood :init)
          (ignore)
  (loop for node in nodelist do (send node :set-neighborhood self))
  (if *hierarchy (send self :show-one-level-derivation)
      (send self :show-neighborhood))
  )

;(Defmethod (neighborhood :show-neighborhood)    BAD CODE
;          (&optional (alu tv:alu-xor))
;  (loop for (one two) in (append nodelist (list (car nodelist)))
;        while two do
;        (send *krwindow :draw-line (fix (send one :xp)) (fix (send one :yp))
;              (fix (send two :xp)) (fix (send two :yp)) alu) ) )

(defMethod (neighborhood :show)
          ()
  "Top-most: uses :show-neighborhood or :show-one-level-derivation."
  ;; (send self :show-neighborhood)
  )

(Defmethod (neighborhood :show-neighborhood)
          (&optional (alu tv:alu-xor))
  (do ((this (append nodelist (list (car nodelist))) (cdr this)))
      ((null (second this)))
    (send *krwindow :draw-line (fix (send (first this) :xp)) (fix (send (first this) :yp))
          (fix (send (second this) :xp)) (fix (send (second this) :yp)) alu))
  )

(Defmethod (neighborhood :show-one-level-derivation)
          (&optional (alu tv:alu-xor;;                tv:alu-seta
                     ))
  (loop for n1 in nodelist
        for n2 in (append (rest1 nodelist) (list (first nodelist)))
```

```
          ;; (cdr (member n1 nodelist))
        do (send *krwindow *relation-drawing-method* (fix (send n1 :xp)) (fix (send n1 :yp))
                 (fix (send n2 :xp)) (fix (send n2 :yp)) alu))

  #|  (dolist (n (append nodelist (list (first nodelist))))
    (send *krwindow :draw-dashed-line (fix (send (first nodelist) :xp)) (fix (send (first nodelist) :yp))
          (fix (send n :xp)) (fix (send n :yp)) alu))|#

(Defmethod (neighborhood :orbit) (&aux r f tf av-xp av-yp theta xp1 xp2 yp1 yp2)
  "Computes *distinct-in-neighborhood and *attract if these flags are set."
  (when *do-orbits
    (Cond-every (*distinct-in-neighborhood
                 (do ((list nodelist (cdr list)))
                     ((< (length list) 2))
                   (do ((b1 (first list)) (b2 (cdr list) (cdr b2)))
                       ((null b2))                ;          (print (send b1 :text))(print (send (first b2) :text))
                     (setq xp1 (send b1 :xp)) (setq xp2 (send (first b2) :xp))
                     (setq yp1 (send b1 :yp)) (setq yp2 (send (first b2) :yp))

                     (if (> (setq r (distance xp1 yp1 xp2 yp2))
                            (* *distinct-delta *char-width))   ;
                         nil                        ;then, no collide
                         (setq f (// *distinct-factor
                                     (expt (if (< r 1.0) 1.0 r) *distinct-r))) ;computer force= r|2 * factor for beyond
                         (setq theta (if (and (equal xp1 xp2) (equal yp1 yp2))
                                         0.0
                                         (atan (abs (- yp2 yp1)) (abs (- xp2 xp1)))))
                         (setq tf (* (expt (string-length (send b1 :text)) *name-length-factor) f *delta-time))
                         (send b1 :add-xa (* tf (cos theta) (if (minusp (- xp1 xp2)) -1. +1.)))       ;text width
                         (send b1 :add-ya (* tf (sin theta) (if (minusp (- yp1 yp2)) -1. +1.)))
                         (setq tf (* (expt (string-length (send (first b2) :text)) *name-length-factor) f *delta-time))
                         (send (first b2) :add-xa (* tf (cos theta) (if (minusp (- xp2 xp1)) -1. +1.)))       ;text width
                         (send (first b2) :add-ya (* tf (sin theta) (if (minusp (- yp2 yp1)) -1. +1.)))
                         ))))

              (*attract  (setq av-xp 0.
                               av-yp 0.)
                         (dolist (node nodelist)      ;compute the average position of all in the neighborhood
                           (setq av-xp (+ av-xp (send node :xp)))
                           (setq av-yp (+ av-yp (send node :yp))))
                         (setq av-xp (// av-xp (length nodelist)))
                         (setq av-yp (// av-yp (length nodelist)))
                                        ;        (send *krwindow :draw-filled-in-circle (fix av-xp) (fix av-yp) 2)
                         (do ((node nodelist (cdr node)) (cx) (cy))
                             ((null node))
                           (setq cx (send (first node) :xp))
                           (setq cy (send (first node) :yp))
                           (if (< (setq r (distance cx cy av-xp av-yp))
                                  (* *attract-delta *char-width))
                               nil
                               (setq f (// (expt (if (< r 1.0) 1.0 r) *attract-r)
                                           *attract-factor))  ;computer force= factor/r|2+1 for beyond
                               (setq theta (if (and (equal cx av-xp) (equal cy av-yp))
                                               0.0
                                               (atan (abs (- av-yp cy)) (abs (- av-xp cx)))))
                               (send (first node) :add-xa (* f (cos theta) *delta-time (if (minusp (- cx av-xp)) +1. -1.)))
```

```
                                (send (first node) :add-ya (* f (sin theta) *delta-time (if (minusp (- cy av-yp)) +1. -1.)))
                                ))
                          )

                (*each-coherence-hierarchy* (each-coherence-hierarchy nodelist))        ;old version, each coh has a head node
                )))

(defun hair-list-hierarchy(&aux (f (* *hair-pull-factor* *delta-time)))
  (loop with neighbor-topics
        for topic in *kr-head-topics* do
        (setq neighbor-topics (send (send topic :neighborhood) :nodelist))
        (send topic :add-ya (minus f))  ;pull up heads
        (send topic :spin)
        (loop for neighbor-topic in neighbor-topics unless (eq neighbor-topic topic) do
              (send topic :add-ya (// f (length neighbor-topics))))))

;;; done by :after below instead
;(defun spun-list-hierarchy(&aux (f (* *hair-pull-factor* *delta-time)))
;  (loop with neighbor-topics
;        for topic in *kr-head-topics* do
;        (setq neighbor-topics (send (send topic :neighborhood) :nodelist))
;        (send topic :add-ya (minus f))  ;pull up heads
;;       (loop for neighbor-topic in neighbor-topics unless (eq neighbor-topic topic) do
;;             (send topic :add-ya (// f (length neighbor-topics))))
;        ))      ; normalized

(defMethod (spun-mixin :spin) ()
  (when *spun-enabled
        (setq theta (mod (+ theta (* *delta-time spun-rate)) (* 2.0 3.14159)))
        (setq spun-radius (* spun-radius *spun-radius-decay))
        (send self :add-xa (* *delta-time (cos theta) spun-radius))
        (send self :add-ya (* *delta-time (sin theta) spun-radius)))  )

(defun each-coherence-hierarchy (nodelist &aux r f)
  (do ((headnode (first nodelist)) (node (cdr nodelist) (cdr node)))
      ((null node))
    (if (> (setq r (distance (send headnode :yp) 0
                             (send (first node) :yp) 0))
           (* *hierarchy-delta *char-width))
        nil                                     ;not too close
        (setq f (// *hierarchy-factor
                    (expt (if (< r 1.0) 1.0 r) *hierarchy-r)))
        (send headnode :add-ya (minus (* f *delta-time)))
        (send (first node) :add-ya (* f *delta-time))
        )
    ))

(Defun startup()
;  (setq *counter (+ 1 *counter))
  (* (- (random *position-random) (// *position-random 2)) *char-width)
  )

;;; ------------------------------------------------------------------ support for naive

(defun fix-heads (list-of-node-names)
  "Insures that hierarchy is done; that *string-topics and *kr-head-nodes are set."
```

```
  (setq *string-topics* list-of-node-names
        user:*hair-list-hierarchy* T)
  )

;;; ---------------------------------------------------------------
(defun kr-window()                      ;:create
  (setq *krwindow (tv:make-window 'my-window
                                  :name "Node Orbit Experiment "
                                  :expose-p t
;                                 :superior (if color-superior-p color:color-screen
;                                               tv:default-screen)
                                  :font-map '(fonts:medfnt fonts:medfnb fonts:cptfont)
        ;;                        :current-font 'fonts:medfnt
                                  :more-p nil))
  (send *krwindow :set-char-aluf tv:alu-xor)

                                                ;(tv:make-blinker *krwindow 'tv:character-blinker
                                                ;         :font 'fonts:bigfnt
                                                ;           :char #\+
                                                ;         :follow-p T
                                                ;         'visibility :blink
                                                ;         )
  )

(defun screen-center(window &aux
                    (x (// (send window :width) 2))
                    (y (// (send window :height) 2)))
  (values x y))
```

# Appendix C
# Short History of THOUGHTSTICKER

## C.1 The first "THOUGHTSTICKER" Software

THOUGHTSTICKER as a software system first existed at the laboratory of System Research Ltd in Richmond-upon-Thames, Surrey, under the direction of Dr Gordon Pask, Research Director. The hardware configuration consisted in a variety of peripheral devices:

1. Text retrieval, in the form of "pigeon hole" slots with printed paper (computer storage was at a premium and was entirely taken up with program);

2. Hi-resolution line graphics displays, for the display of the evolving knowledge representation, especially the verification of valid constructions;

3. Manual, paste-up board for user construction of complete knowledge "maps", with computer-controlled status indicators;

4. A digital computer, the heart of the system, running the software of the THOUGHTSTICKER system.

The software for this original system was written in a LISP variant. Both THOUGHTSTICKER and the LISP-like language it was coded in, called LISPN, was written by Robert Newton of the staff of System Research Ltd. LISPN was an interpreter written in assembly code for the Computer Automation LSI machine, one of the first mini-computers

available in England. LISPN contained many of the primitives of the LISP 1.5 of McCarthy, but specifically not the lambda operator for function definitions. This would seem a fundamental contradiction to LISP itself, and it was, as insisted on by Pask. One justification of this was to avoid any observer's confusion over the intention of the implementation, and as a snub of the lambda calculus itself, which Pask felt was an unnecessary trivialisation of Currey and Pheys combinator logics. This THOUGHTSTICKER was limited in complexity of data structure and in performance. All system code was swapped from 8 inch floppy disk, rather inefficiently. The system as whole with its peripherals was a *tour de force*, considering the era of computing and the circumstances of funding. One of its major contributions was its originality: it was one of the first and perhaps the first truly domain-free software programs for knowledge representation. It could represent the knowledge and beliefs of any individual, in any domain, using the same software. It also was the first program to attempt to display in a useful and psychologically meaningful way the topological structure of the knowledge.

## C.2 The first micro-based implementation: MTHSTR

The limitations of the original THOUGHTSTICKER were clear to anyone viewing it, and the primary one was perhaps its imprisonment in a completely non-standard, baroque and incompatible environment of the research laboratory (literally, and often referred by those who visited, as the research basement) of System Research. The obvious path for future exposure and ultimately extension was to re-implement the system in some compatible and reasonably available micro-processor environment. For the moment, the limitations of memory size implicit in micro computers would be set aside for the other benefits.

Robin McKinnon-Wood, long-time collaborator of Pask and contributor to the Cambridge University Language Research Unit, conceived of and coded a database scheme for the BASIC language. The hardware was produced by Research Machines Limited (RML), and ran a Z80 processor in a custom operating system. This was in 1978-79 and CP/M was not yet widely available, at least on machines distributed in the UK. The BASIC was a fairly good implementation of what would become the MicroSoft version of the language. This scheme was clever in that it used the string capabilities to emulate what would be simple list-processing primitives in LISP. In some 6 pages of transferable code, MTHSTR, as it was dubbed, contained the basic operations of Lp; specifically, instatement of coherences, Genoa contradiction checking (for certain cases only), Prune and Selective Prune (for most cases but none of great depth), and other utilities such as listing topics, coherences and pruning structures. McKinnon-Wood, an algorithm man from way back, understood the meaning of Turing computable and universal machines. The cleverness was performing complex, multi-dimensional parsing using a lexical scheme and one-dimensional string arrays. The means for representing the Lp structure of a coherence was arrived at in collaboration Pangaro. This involved eliminating the redundant storage of each pruning of each coherence, and instead storing a single cluster in which the prunings would be unfolded. Some further experiments were done in APL and a "structural" (*i.e.* macro-level) condense/expand was written in MacLISP by Pangaro but none of these efforts produced a complete THOUGHT-STICKER.

## C.3 PASCAL TSTIK

The Applied Psychology Unit (APU) of AMTE Teddington (now named the Behavioural Science Division of ARE Teddington) became interested in Pask's work and funded an effort to place THOUGHTSTICKER into

the HP1000 systems which they had on-site at APU. Pangaro, initially with the aid of McKinnon-Wood and the MTHSTR code, provided written specifications for the database routines and Lp operations, including a crude notational approximation of condense/expand as extracted from long arguments with Pask himself as to their nature and significance. These specifications were given to a programmer, working for an established Ministry of Defense contractor. The system was later taken over by Peter Clark, a computer scientist and student of Pask who endeavored to finish it. The entire project was fraught with difficulties, including but not limited to difficulties with using contractors not familiar with CT, obsolete operating systems, poor management of computing facilities, and Collapsing research organizations. Despite this, TSTIK, as it was called, contained a menu of some twenty commands and some features only recently duplicated in significantly improved circumstances. Included in TSTIK were instatement of coherences, full Genoa contradiction check, pruning and selective pruning, saturation, condensation/expansion (crude), bifurcation of topics, and merging of universes. It was used for a brief time in an exploratory way but was lost when the computing environment in which it ran was superseded.

## C.4 Apple CASTE, a version of THOUGHTSTICKER

MTHSTR itself as a historical artifact first demonstrated THOUGHTSTICKER functions in a micro environment. More importantly, its database scheme became the basis for a system with many cosmetic and functional extensions, running in the Apple II microcomputer. For the record, it should be noted that the Apple configuration required to run this version of THOUGHTSTICKER is not a pure Apple system: it requires a Z80 processor card, 80-column text display card, the CP/M operating system, and, to run efficiently on

reasonably-sized demonstration environments, 256K RAM-Disk extender cards.

These extensions, and ultimately a complete re-write and major expansion of MTHSTR, were made by Pangaro under contract to APU. (Later, Scott Henderson of the staff of PANGARO Incorporated made some further changes.) The system was initially called CASTE, to emphasize its authoring and tutorial capabilities. Perforce it contains THOUGHTSTICKER elements, especially instatement of coherences, full Genoa contradiction checking, pruning and selective pruning, and saturation. The primary contribution of this version was its ease of use, the existence of its usable documentation, and the fact that it proved the viability of the approach in micro computers.

One innovation of the system, conceived in collaboration by Dik Gregory of APU and Pangaro, was the use of sentences of English language text to contain concepts in the system. The PASCAL TSTIK also contained this feature at one point, but the full development was done in the BASIC version. These sentences would be provided by the author and be the primary information seen by the student while being tutored.

A "Rules Tutor" for the naval simulation game called HUNKS was constructed by Pangaro in the Apple. This game involves opposing commanders of fleets of vessels and the ability to command the simulation to move the vessels, fire missiles, and call for information on graphics displays. Gregory used the system to construct both text and simple graphics frames to provide tutorials for learners of rules of the game. The configuration required a second Apple II, running the HUNKS code and driven by the CASTE software in the other machine. The remote machine was thus a graphics engine for the representation of tutorial material and game situations.

A variety of test domains have been constructed for this implementation, including a word processing tutor (which contains basic concepts of word processing as well as command conventions and user "help features") and a database for corporate documentation. This Apple system written in Microsoft BASIC then became the basis of a multi-Apple version developed under the direction of Pask while under contract to the Army Research Institute (ARI), Virginia. Pask and his associates at the Centre for System Research and Knowledge Engineering, Concordia University, Montreal, Canada, added additional controllers for slide projections and multiple-screen tutorial modes, and made certain functions (notably prune) more robust.

## C.5 THOUGHTSTICKER 3600

The purpose of developing the Apple CASTE version of THOUGHT-STICKER was to bridge the time between the demise of the PASCAL version and the planned version to be written in LISP on a hardware configuration of sufficient size and performance to thoroughly test the capabilities of THOUGHTSTICKER in a serious research implementation. APU placed a contract in early 1982 with PANGARO Incorporated to obtain the necessary hardware and expertise in Washington DC, while themselves beginning the procurement process for their own, identical hardware.

The hardware chosen by Colin Sheppard of APU was the Symbolics 3600, a special-purpose mini-computer which is a hardware engine for running ZetaLISP, a superset of MacLISP, itself derived from McCarthy's LISP 1.5. The Symbolics system is unsurpassed in performance, features and software support. It arrived at PANGARO Incorporated in Washington DC in April of 1983.

Over the course of 18 months, a THOUGHTSTICKER was developed using the object-oriented programming techniques of the Symbolics called Flavors. A set of user interface windows were developed, along with the underlying code, by Jeffrey Nicoll of PANGARO Incorporated. This THOUGHTSTICKER has the functions of Lp up to but not including "condense/expand" although the primitives required for it are present. The emphasis is on displaying the details of the evolving knowledge structure and maximizing the choices allowed to the user at any time. This was an explicit choice, in order to study the implications of all of the Lp operations under the pressure of practical use. It is necessary therefore to have some familiarity with CT and Lp in order to use these user interfaces to this version of THOUGHTSTICKER, called the "research implementation." Information basic to its use is provided in the THOUGHTSTICKER User Manual, available as an Annex to this dissertation.

Unlike any other implementations, or any considerations given by the underlying theory, this version of THOUGHTSTICKER as conceived by Pangaro and Nicoll and written by Nicoll is centrally concerned with the differentiation between "P-Individuals", who might be different individual users or the same user under differing circumstances of goals, needs, or occasions. Conflict resolution therefore includes tools for declaring existing assertions accepted or not accepted relative to the current "contexture", alias "persona" (as a P-Individual is called in the system).

## C.6 "Naive" THOUGHTSTICKER

A further interface was started in 1984, originally as a request of ARI, which would attempt to allow THOUGHTSTICKER to be used by individuals not at all familiar with CT or Lp operations. This version,

dubbed "Naive THOUGHTSTICKER" was coded by Pangaro under the specific design constraint that choices not be offered as "pop up" menus, and that a linear set of questions be asked of the user in order to perform all functions. The result is an excellent tutorial environment. Authoring is less successful, in that the implications of choices made by the naive author cannot be seen until some familiarity is gained, and the variety of options available for conflict resolution make it difficult to be patient with the linear resolution questions when the desired change is known beforehand. The experience gained in writing both the research implementation and the Naive interface is being used in a complete re-writing of the system as the basis of commercial versions of intelligent training software in the Symbolics.

## C.7 The Expertise Tutor

THOUGHTSTICKER has also been integrated into a complete (prototype) system for training and job-aiding called the Expertise Tutor (the XT), as an interpretation by Pangaro of Colin Sheppard's original concept of Intelligent Support Systems. The XT provides not only the "descriptive" elements of how to play the HUNKS naval simulation game (such as the mission, number and types of vessels, playing rules, *etc.*) but also the "prescriptive" elements such as how to formulate a correct command, what strategies might be employed and what conditions of the game should be attended to at any given time.

THOUGHTSTICKER provides the descriptive elements and the Naive interface is integrated into the XT user window. The "prescriptive" elements are provided by a system called the SEEKER, conceived and written by Scott Henderson of PANGARO Incorporated and based on a prescriptive interpretation of entailment meshes as implied originally by Pask and interpreted by Pangaro. The SEEKER was a means for referring

to HUNKS objects and the status of the HUNKS game, while linking these references into sentence-like tactics that would produce commands for the HUNKS game. Thus a SEEKER-based computer player was constructed by Henderson, as an added bonus to the SEEKER concept. The original purpose was only to provide a means for presenting prescriptive knowledge to a learner and allowing the definition of new tactics on-the-fly in the course of running the simulation.

Thus knowledge of the game is acquired by the user both descriptively and prescriptively, and the user could at any point choose which mode to operate in. The underlying databases were linked (although they could have been and perhaps should have been the same database, but this was impractical given the development strategy of the project; however the effect for the user was the same without noticeable loss of efficiency). With the addition of the HUNKS simulation itself, the complete user interface consisted of the explicit differentiation between the levels of interaction between learner and teacher as set out in Conversation Theory. As designed by Pangaro, this is the first embodiment of the complete conversational structure in software.

## C.8 Do-What-Do

A few years before the XT was constructed, Pangaro had proposed an interface concept called "Do-What-Do" in which the distinction between descriptive and prescriptive intention on the part of the user was both made explicit and made available to the user in software. All user interface interaction was seen as either question ("What is this object in the interface...") or command ("Perform this action..." or "This is the object I mean..."). Thus an interface would be completely accessible to any user by means of these two basic commands, and all of the user modelling features of THOUGHTSTICKER (such as the history of

shared vocabulary and common context) would continue to tune the choices made by the software.

When the XT described in the previous section was built, the "Do-What-Do" concept was implemented in a basic form. The "Do" and "What" distinction was implemented by two distinct buttons on the mouse. Thus "Click-Left" meant "Do" and "Click-Right" meant "What." Results were favorable though the research programme did not allow for any explicit exploration or further development of this feature.

## C.9 Interactive Videodisc Interface

In 1986, the US ARI contracted with PANGARO Incorporated to connect an interactive videodisc interface to THOUGHTSTICKER. Although the direct purpose was to allow for experiments to be performed comparing THOUGHTSTICKER with other forms of training (including platform, alias classroom, instruction, and conventional computer-aided instruction) the facility provided was a generalized one. Thus when a user is provided with a model of a topic or relation, a videodisc still or sequence with optional audio channel(s) is shown. These are chosen by the expert in the authoring mode with a basic facility for attaching videodisc models to the THOUGHTSTICKER database.

# Appendix D
# Bibliography


Bannister, D. & Mair, J. M. M. (1968): *The Evaluation of Personal Constructs*. Academic Press, New York.

Barr, Avron & Feigenbaum, Edward A. (1981): *The Handbook of Artificial Intelligence*, Volume I, William Kaufmann, Inc., Los Altos, California.

Bateson, Gregory (1960): *Steps To An Ecology of Mind*. Ballantine Books, New York.

Bogner, Marilyn Sue (1986): "Course Assembly System and Tutorial Environment: An Evaluation." Proceedings of the National Educational Computing Conference, San Diego, California, June 1986.

Brachman, Ronald J. (1979): "On the Epistemological Status of Semantic Networks." In N. V. Findler (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, New York.

Brachman, Ronald J. (1985): " 'I Lied about the Trees' Or Defaults and Definitions in Knowledge Representation." In *The AI Magazine*, Fall 1985 Issue, Menlo Park, California.

Clark, Peter (1980): "Saturation, Bifurcation and Analogy in Lp: A Position Paper." Presented at the Third Richmond Conference in Decision Making, Richmond, Surrey, 1980.

Chomsky, N. (1968): "Formal Properties of Grammars." In *Handbook of Mathematical Psychology*, Volume II, Chapter 12, John Wiley, New York.

Deutsch, David (1985); "Quantum computability." Royal Society, London.

Dreyfuss, Hubert & Dreyfuss, Stuart (1986): "Why Computers May Never Think Like People." In *Technology Review*, January 1986, MIT, Cambridge, Massachusetts.

Duda, Richard O. & Shortcliffe, Edward H. (1983): "Expert Systems Research." In *Science*, Volume 220, Number 4594, 15 April 1983.

Feigenbaum, E. A. & Feldman, J. (1963): *Computers and Thought.* McGraw Hill, New York.

Feigenbaum, E. A. & McCorduck, P. (1983): *Fifth Generation.* Addison-Wesley, Menlo Park, California.

Gregory, Richard (Dik) (1981): "Genoa Rules...OK?" Technical Memorandum, Admiralty Marine Technology Establishment, E1/P4.3/215/82, Teddington, Middlesex.

Hewitt, Carl (1972): "Procedural Semantics: Models of Procedures and Teaching of Procedures." In Randall Rustin (Ed.), *Natural Language Processing*, Algorithmics Press, 1972.

Hewitt, Carl & Baker, H. (1977): "Laws for Communicating Parallel Processes." In *Proceedings of IFIP Congress 77* Toronto, August 1977.

Hillis, W. Daniel (1985): *The Connection Machine*. The MIT Press, Cambridge, Massachusetts.

Humphreys, Patrick & Wisudha, A. (1975): "MAUD: -- An interactive computer program for the structuring, decomposition and recomposition of preferences between multiattributed alternatives." Technical Report 79-2, Decision Analysis Unit, Brunel University, Uxbridge, Middlesex.

Humphreys, Patrick & McFadden, Wendy (1980): "Experiences with Maud: Aiding the Decision Structuring versus Bootstrapping the Decision Maker." In *Acta Psychologica* 45, North Holland Publishing Company, Holland.

Kearsley, Greg P. (1977): "Some Conceptual Issues in Computer-Assisted Instruction." In *Journal of Computer-Based Instruction*, Volume 4 Number 1.

Kelly, G. A. (1966): "A Brief Introduction to Personal Construct Theory." In D. Bannister (Ed.), *Perspectives in Personal Construct Theory*, Academic Press, London and New York.

Laing, R. D., Phillipson, H., & Lee, A. R. (1966): *Interpersonal Perception*, Tavistock Publications, London.

Lettvin, J. Y. (1985): "Warren McCulloch and the Origins of AI." Reprinted in Trachtman, P (Ed.), *Cybernetic*, publication of the American Society for Cybernetics, Volume 1, Number 1, Summer-Fall 1985.

Marante, G. J. & Laurillard, D. M. (1981): "A View of Computer Assisted Learning in the Light of Conversation Theory." Institute for Educational Technology, University of Surrey.

Maturana, Humberto (1978): "Biology of Language: The Epistemology of Reality." In Miller, George A. & Lenneberg, Elizabeth (Eds.), *Psychology and Biology of Language and Thought.*

Maturana, Humberto (1982): "What is it to see?" Reprinted in Trachtman, P (Ed.), *Cybernetic*, publication of the American Society for Cybernetics.

Maturana, Humberto (1986), in Pedretti, A. *et al* (Eds.), *Conversations in Cybernetics*, Princelet Editions, London.

Michie, Donald (Ed.) (1982): *Introductory Readings in Expert Systems.* Gordon and Breach, Science Publishers, Inc., New York.

Minsky, M. (1967): *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey.

Minsky, M. (Ed.) (1968): *Semantic Information Processing.* MIT Press, Cambridge, Massachusetts.

Minsky, M. (1975): "A Framework for Representing Knowledge." In P. H. Winston (Ed.), *The Psychology of Computer Vision*, McGraw Hill, New York.

Minsky, M. (1986): *Society of Minds.* Simon and Schuster, New York.

Negroponte, N. (Ed.) (1977): *Graphical Conversation Theory*, Proposal of the Architecture Machine Group, MIT, to the National Science Foundation, 1977.

Nicoll, Jeffrey F. (1985): "To be or not to be." Technical Memorandum, PANGARO Incorporated, June 1985.

Nilsson, Nils J. (1980): *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, California.

Pangaro, P. (1980): "Programming and Animating on the Same Screen at the Same Time." *Creative Computing*, Volume 6, Number 11, November 1980.

Pangaro, P. (1982): "Beyond Menus: The Ratzastratz or the Bahdeens." Proceedings of the Harvard Graphics Conference, 1982.

Pangaro, P., Steinberg, S., Davis, J. & McCann, B. (1977): "EOM: A Graphically-Scripted, Simulation-Based Animation System." Memorandum, Architecture Machine Group, MIT, August 1977.

Pangaro, P. & Harney, H. (1983): "CASTE: A System for Authoring and Tutoring." *Data Training*.

Pangaro and Nicoll (1983), "Deleting the Knowledge Engineer: A practical implementation of Pask's protologic, Lp." Based on a paper presented at the Seminar on Machine Intelligence in Defence, Ministry of Defence, Weymouth, Dorset, UK.

Pangaro, P. (Ed.) (1985): "THOUGHTSTICKER User Manual." PANGARO Incorporated. Attached to this dissertation as an Annex.

Pask, G. (1958): "Physical Analogues to the Growth of a Concept." Proceedings of Symposium on Thought Processes, National Physical Laboratory, Teddington, Middlesex, November 1958.

Pask, G. (1963): "The meaning of cybernetics in the behavioural sciences." In *Cybernetics of Cybernetics*, published by the Biological Computing Laboratory, University of Chicago, Urbana, Illinois.

Pask, G. (1972): "Anti-Hodmanship: A Report on the State and Prospects of CAI." In *Programmed Learning and Educational Technology*, Volume 9 Number 5, September 1972.

Pask, G. (1975a): *Conversation, Cognition and Learning*. Elsevier Scientific Publishing Company, Amsterdam.

Pask, G. (1975b): "Aspects of Machine Intelligence." In Negroponte, N. (Ed.) *Soft Architecture Machines*. MIT Press, Cambridge, Massachusetts.

Pask, G. (1975c): *The Cybernetics of Human Learning & Performance*. Hutchinson Educational, London.

Pask, G. (1976a): *Conversation Theory: Applications in Education and Epistemology*. Elsevier/North Holland Inc., New York.

Pask, G. (1976b): "An Outline Theory of Media for Education and Entertainment." Proceedings of Third European Meeting on Cybernetics and System Research, Vienna 1976.

Pask, G. (1976c): "Revisions in the Foundation of Cybernetics and General Systems Theory as a Result of Research in Education, Epistemology and Innovation (Mostly in Man-Machine Systems)." In Proceedings of 8th International Congress on Cybernetics, Namur, Belgium.

Pask, G (1978): "The protologic Lp." Technical Memorandum of System Research Ltd, Richmond, Surrey.

Pask, G. (1979): "Against Conferences or the Poverty of Reduction in SOP-Science and POP-Systems." In Proceedings of Society for General Systems Research, London, 1979.

Pask, G. (1980a): "Developments in Conversation Theory." In *International Journal of Man-Machine Studies*, Volume 13.

Pask, G. (1980b): "The Limits of Togetherness." In Proceedings of IFIPS 80, 1980.

Pask, G. (1980c): "Organizational Closure of Potentially Conscious Systems." In Zelany, M., *Autopoesis*, Elsevier/North Holland Inc., New York.

Pask, G. (1980d): "An Essay on the Kinetics of Language." In *Ars Semiotica*, Volume III:1.

Pask, G. (1983): "THOUGHTSTICKER Developments: A Concept Paper." Centre for System Research and Knowledge Engineering, Concordia University.

Quillian, M. R. (1968): "Semantic Memory." In Minsky, M. (Ed.) (1968), *Semantic Information Processing*. MIT Press, Cambridge, Massachusetts.

Shannon, Claude E. & Weaver, Warren (1964): *The Mathematical Theory of Communication*. University of Illinois, Urbana, Illinois.

Shank, R. C. (1980): "Language and Memory." In *Cognitive Science*, Volume 4.

Shank, R. C. & Abelson, R. P. (1975): "Scripts, plans and knowledge." Proceedings of the Fourth Joint International Conference on Artificial Intelligence, Tbilisi.

Shaw, Mildred (1980): *On Becoming a Personal Scientist*. Academic Press, London.

Sheppard, Colin, "Man-Computer Studies Section: Rationale and Work Programme." Applied Psychology Unit, Admiralty Marine Technology Establishment, Teddington, England, 1981.

Varela, Francisco (1975): "A Calculus for Self-Reference." In *International Journal of General Systems*, Gordon and Breach Science Publishers, Ltd., Volume 2.

von Foerster, Heinz (1960): "On Self-Organizing Systems and their Environments." In Yovits, M. C. & Cameron, S. (Eds.), *Self-Organizing Systems*, Pergamon Press, London.

von Foerster, Heinz (1977): "Objects: Tokens for (Eigen-) Behaviors." In Inhelder, B., Garcia, R., & J. Voneche (Eds.), *Homage a Jean Piaget: Epistemologie genetique et equilibration*, Delachaux et Niestel, Neuchatel.

von Foerster, Heinz (1980): "Cibernetica ed Epistemologia." In Bocchi,

Gianluca & Ceruti, Mauro (Eds.), *La sfida della complessita*, Feltrinelli, Milano.

von Wright, G. H. (1963): *Norm and Action*, Kegan Paul, New York.

Winograd, T. & Flores, F. (1986): *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing, Norwood, New Jersey.

# Appendix E
# Glossary of Terms

This Glossary defines terms from a perspective consistent with the basis of this dissertation, rather than common usage.

**Adicity:** The number of elements in a relation. For example, a coherence with three topics has an adicity of three. The term is a form of "-adic" as for example, a triadic (three element) relation.

**Agreement:** Declaration made by an observer about transactions witnessed across a distinction defining P-Individuals, based on the assertion by each of the individuals that the others' derivation and use of a concept is consistent with their own. See Understanding.

**Analogy:** A relation in Lp associating topics by declaring their similarities and differences.

**Author:** A term from computer-based instruction, the author is the creator of the subject matter and usually a teacher or subject matter expert. In THOUGHTSTICKER however, there are situations where the knowledgebase needs to be extended locally by users, in order to customize it to local practices or needs, or to expand its contents. In this situation, the user, even possibly the learner, may take the role of author; the system maintains the identity of each author and hence can control access to the material as appropriate.

**Authoring or To Author:** The act of creating the subject matter.

**Authoring Module:** That part of the THOUGHTSTICKER systems which allows a user to create or extend the knowledgebase. See Author, above.

**Bifurcation:** The splitting of a topic into two or more topics, creating distinctions which keep the topics differentiated.

**Branching:** This is the primary format for conventional computer-aided instruction. The subject matter is arranged in a "tree" of fixed routes which the learner follows. The results of tests of the learner determine "branching" through the structure. THOUGHT-STICKER removes the restrictions of branching by utilizing a knowledge representation for the subject matter, freeing the learner to explore the subject based on individual experience, needs, and conceptual style.
THOUGHTSTICKER can also be used in a sequential mode for convenient incorporation of existing computer-based instructional material, but still with liberal opportunities for the learner to be driven by uncertainty and curiosity.

**Browser:** That part of THOUGHTSTICKER that allows inspecting the knowledgebase. This term reflects the application of THOUGHT-STICKER to information management, where the user is accessing, rather than extending, the knowledgebase.

**CASTE:** The Course Assembly System and Tutorial Environment was an electro-mechanical system built to facilitate the construction of courseware, in an era when hardware was too expensive to also conceive that delivery of training could be done by a related system. The basis by which it structured the subject matter was a forerunner to THOUGHT-STICKER.

**CBT/CAI:** Computer-Based Training and Computer-Aided Instruction are terms from the training industry. The former is concerned with all aspects of training whether pedagogical or not, including managing the records of a student population and tracking the progress of individual students. The latter emphasizes the pedagogical component, especially in

how the delivery of training *via* computer can be made different than other forms of training (classroom, self-study from books). THOUGHTSTICKER goes far beyond existing systems in the effectiveness of its pedagogical techniques and it maintains a complete knowledgebase of the learner's progress which is available for use at all times during the interaction.

**Coherence:** An Lp relation among topics, in which any topic (isolated for the purpose) may be reproduced or reconstructed from the remainder. This affords a high degree of redundancy, the basic unit of memory, and the basis from which conflict detection and resolution emerges.

**Communication:** Transactions between individuals that involves transfer of previously-agreed symbols; no new symbols may be transferred. Communication implies transfer of data about transmitted state(s) as related to a known class of possible states. See Conversation.

**Conflict:** The interaction of processes whereby simultaneous execution cannot persist without either one or more processes being destroyed or being resolved (*via* structural changes in the processes).

**Conflict Detection:** A software simulation of concurrent processes whereby conflicts among relations are computed.

**Conflict Resolution:** A procedure as guided by macro software whereby conflicting relations are modified to eliminate conflict.

**Conversation:** A term from CT for generalized interaction between P-Individuals which consists of transactions in an agreed language designated "L" and taking place on multiple "levels" as defined by an observer. The transactions may be "I-you" referenced, insofar as they are held across the distinction between P-Individuals at the same level; or "it" referenced insofar as one individual treats the other as its environment and does not allow, but rather insists on, response.

**Conversation Theory or CT:** The name given to a theory of individuals and cognition.

**Courseware:** The subject matter as created by the author, represented in the training software, and delivered to the student. It may consist of any combination of text, graphics, simulation, and videodisc media.

**Cybernetics:** The study of systems from their information and relativistic (observer-bound) basis. See Second-order Cybernetics.

**Do-What-Do:** A conceptual approach to MMI, and an implementation in the system called the Expertise Tutor. Given the requirements for any transaction language to be capable of question and command, the most direct explanation to a user of a software interface is to consider the basic interactions as either a question ("What is this [element of the screen]?") or a command ("Execute this command!"). This is the What and Do portion of the term. The additional Do at the end, completing Do-What-Do, emphasizes the iterative nature of the entire process and also how merely asking is not sufficient; some execution is necessary for comprehension of the environment. This approach was implemented by dedicating 2 of the 3 mouse buttons of the Symbolics machine to Do and What. The third, middle button was then used for a global orientation. but not fully integrated into the scheme (as for example, a "Why" button might be).

**Expertise Tutor: See Do-What-Do.**

**Frames** (from AI): Data structures intended as models of human thought, were slots and their values stand for memories. "Default values" supply information when specific experience has not provided any.

**Frames** (from Computer-Aided Instruction): The fundamental unit of experience for the learner in computer-aided instruction. Usually text or

some combination of text and graphics, frames are linked in a fixed structure within which the learner "branches." Frames are a gross accumulation of many features of the knowledge to be learned which are neither broken down by the author nor distinguishable by the system. Hence a CAI system cannot learn very much about the individual learner, nor individualize its interaction very much, because it can distinguish very little of the learner's attention or uncertainty.

**Individual:** See P-Individual.

**Knowledge Representation** or **Knowledgebase:** Term from the field of artificial intelligence referring to a software structure that is intended to represent knowledge, often of an expert. A number of approaches exist and none are considered to solve the problem for the general case. THOUGHTSTICKER uses a knowledge representation that was developed from a cognitive theory especially for the purpose of representing knowledge to be learned. Hence the heuristics that operate on the knowledge structure are well-suited to variations in conceptual style and provide highly individualized interaction for each user.

**Knowledge Elicitation:** The process of extracting information from a human, usually for the purpose of then placing it into a knowledge representation.

**Narrative:** The term for a THOUGHTSTICKER sequence that tells a story and has some of its import conveyed by the particular order it is seen in; hence its name. The THOUGHTSTICKER features which are concerned with providing conventional computer-based instruction as part of THOUGHTSTICKER use the Narrative facilities for implementing a frame sequence. The learner is still free to explore away from the Narrative and to return easily to it as desired.

**Language** or **L:** A transaction medium capable of question and command as well as description and predication.

**Lp:** One example of a protologic or protolanguage. Proto, meaning "below", is used to indicate that Lp is not itself a language or logic; it is a substrate on which one might be built. See the glossary entry Language.

**Learner:** In computer-based instruction, this term takes on its usual meaning. However, the learner may be focused on the subject matter for a variety of purposes: to cover the entire subject to acquire general skills; to perform a specific task requiring a subset of skills; or to answer a specific question or perform a single operation. THOUGHTSTICKER reacts differently for each of these purposes (see Persona).

**Macro Software:** Simulation at a level relative to atoms of the knowledgebase, such that the elements taken as atoms (which normally have further structure of sub-functions as indicated by the theory) are taken as static. Thus in the case of CT, topics are considered as static nodes in the knowledgebase. See Micro Software.

**Man-Machine Interface** or **MMI:** The user experience at the computer console. In context, MMI may also refer to the software required to implement a particular user experience at the interface to the software.

**Micro Software:** Simulation at a level where atoms from the theory are the basic units of the simulation, rather than some "higher" level. Thus in the case of CT, topics are interpreted as dynamic processes each with individual interactions acting upon it.

**P-Individual:** For "psychological individual", a conceptual entity that is distinguished by an observer based on criteria of differences as defined by the observer, between or across conceptual points of view rather than physical boundaries. Hence, a single individual "person" consists of many

(and often conflicting) perspectives; alternatively, many persons can make up a group, as for example in religion or politics, and be the "same individual" so far as a particular set of their beliefs is concerned.

**Persona:** A term unique to THOUGHTSTICKER, the persona refers to (1) the individual currently using the system, and (2) the current goal of that use (see Learner, above). The persona can be used by THOUGHTSTICKER to guide its heuristics and present information in an adaptive way for this individual with a current goal. The user, whether author or learner, is identified to the system, and all history is attached to, the persona.

**Relations:** Structural associations between elements, usually topics, in the knowledgebase. See Coherence, and Analogy.

**Rule of Genoa:** The name given by Pask to CT's bifurcation principle, named after the city of Genoa where resided Vittorio Midoro, who asked a question about representing coherence and analogy in the same diagram and thereby provided a hint as to the creation the principle.

**Saturation:** As an Lp process: existing topics are joined to others in coherences. As a THOUGHTSTICKER function: new coherences, which do not conflict with existing relations, are proposed to the author for possible instatement.

**Second-order Cybernetics:** The later development of cybernetics which emphasizes certain systems' capability to refer to themselves, *i.e.* to model their own behavior or the behavior of others. The full implications of the subjective nature of experience appears in second-order.

**THOUGHTSTICKER:** Macro software based on CT and its knowledge calculus, Lp.

**Topic:** Minimal, atomic unit of a THOUGHTSTICKER knowledgebase. Simulated in macro software as a static node, the topics of CT are dynamic repertoires of processes which converge in value, as do Eigen values. Achieved in micro software by an approach where each topic is a process.

**Tutoring Module:** That part of the THOUGHTSTICKER system which allows a user to explore the knowledgebase. This term reflects the application of THOUGHTSTICKER to training, where the user is in the role of student.

**Understanding:** P-Individuals hold agreements over understandings. See Agreement.

Write Watcher 1: Cybernetics-and-all-that in context of Holist

Statement editing commands:

| | | | |
|---|---|---|---|
| Environment | Add topic | Clear | What was |
| Retrieve lost work | Kill if not used | Use text topics | Instate |
| Choose Linearizing Analogy | Choose Linearizing Construct | | |

Topics:

To represent knowledge is the goal of artificial intelligence programming.

END-key displays topics.

To represent knowledge is the goal of artificial intelligence programming.

Select to show menu of choices
81/12/07 12:55:14 pen LP: fyi

Figure 1

Write Watcher 1: Cybernetics-and-all-that in context of Holist

Statement editing commands:

| | | | |
|---|---|---|---|
| Environment | Add topic | Clear | What was |
| Retrieve lost work | Kill if not used | Use text topics | Instate |
| Choose Linearizing Analogy | Choose Linearizing Construct | | |

Topics: artificial intelligence goals knowledge

To represent knowledge is the goal of artificial intelligence programming.

END-key displays topics.

To represent knowledge is the goal of artificial intelligence programming.

< artificial intelligence ^ goals ^ knowledge > Cannot be entered legally as a coherence.

Try to resolve conflicts

Make this proposal an analogy

Use model(s) of proposal as model(s) of topics

Make model(s) independent of proposal

Choose bold item

Examine the conflicts of this proposal in order to determine a resolution

01/12/87 13:02:30 pan LP: fyi

Figure 2

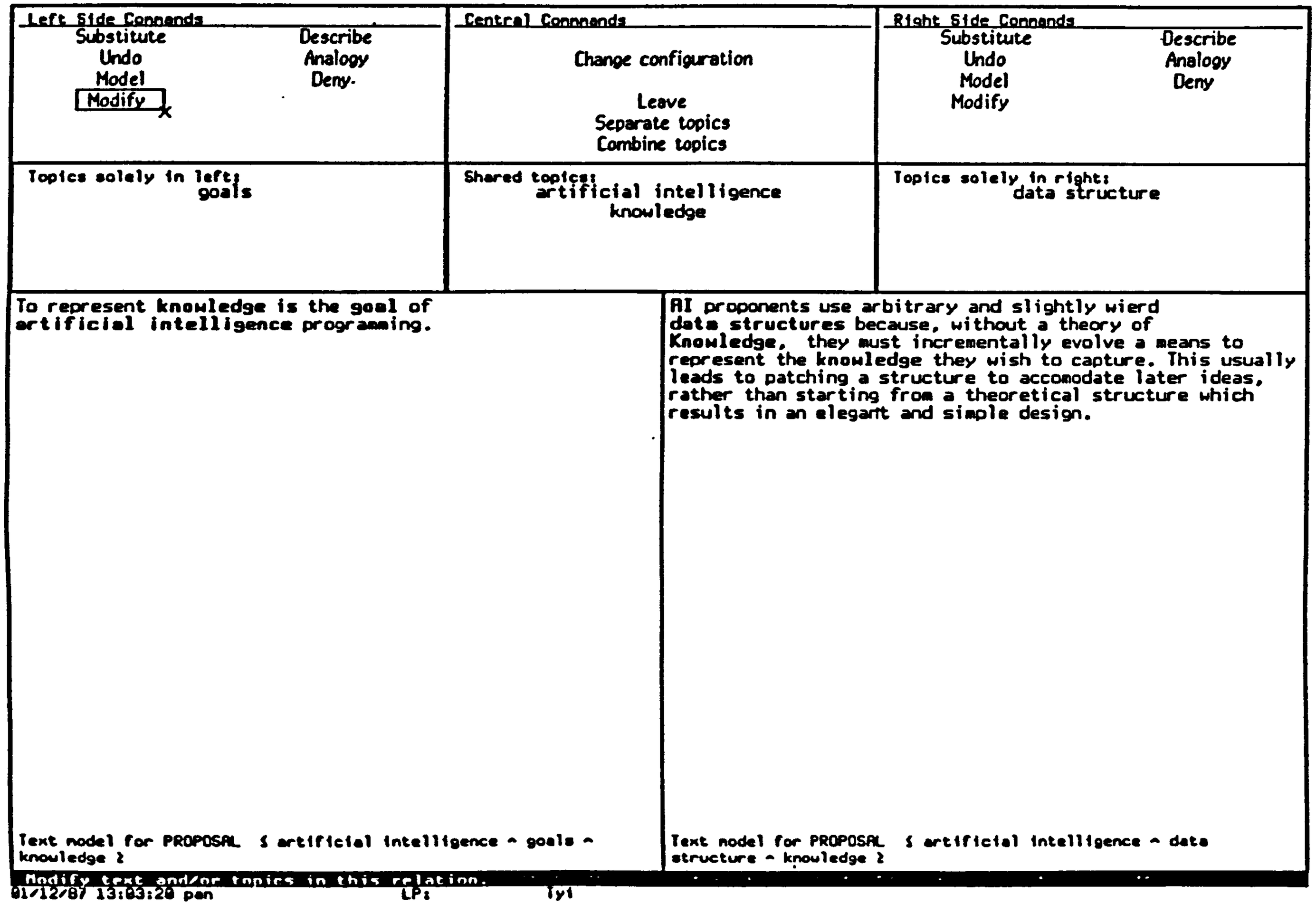

Left Side Commands
Substitute
Undo
Model
Modify
Describe
Analogy
Deny.
Central Commands
Change configuration
Leave
Separate topics
Combine topics
Right Side Commands
Substitute
Undo
Model
Modify
Describe
Analogy
Deny
Topics solely in left:
goals
Shared topics:
artificial intelligence
knowledge
Topics solely in right:
data structure
To represent knowledge is the goal of artificial intelligence programming.
AI proponents use arbitrary and slightly wierd data structures because, without a theory of Knowledge, they must incrementally evolve a means to represent the knowledge they wish to capture. This usually leads to patching a structure to accomodate later ideas, rather than starting from a theoretical structure which results in an elegant and simple design.
Text model for PROPOSAL { artificial intelligence ^ goals ^ knowledge }
Text model for PROPOSAL { artificial intelligence ^ data structure ^ knowledge }
Modify text and/or topics in this relation.
01/12/87 13:03:20 pan
LP:
fyi

Figure 3

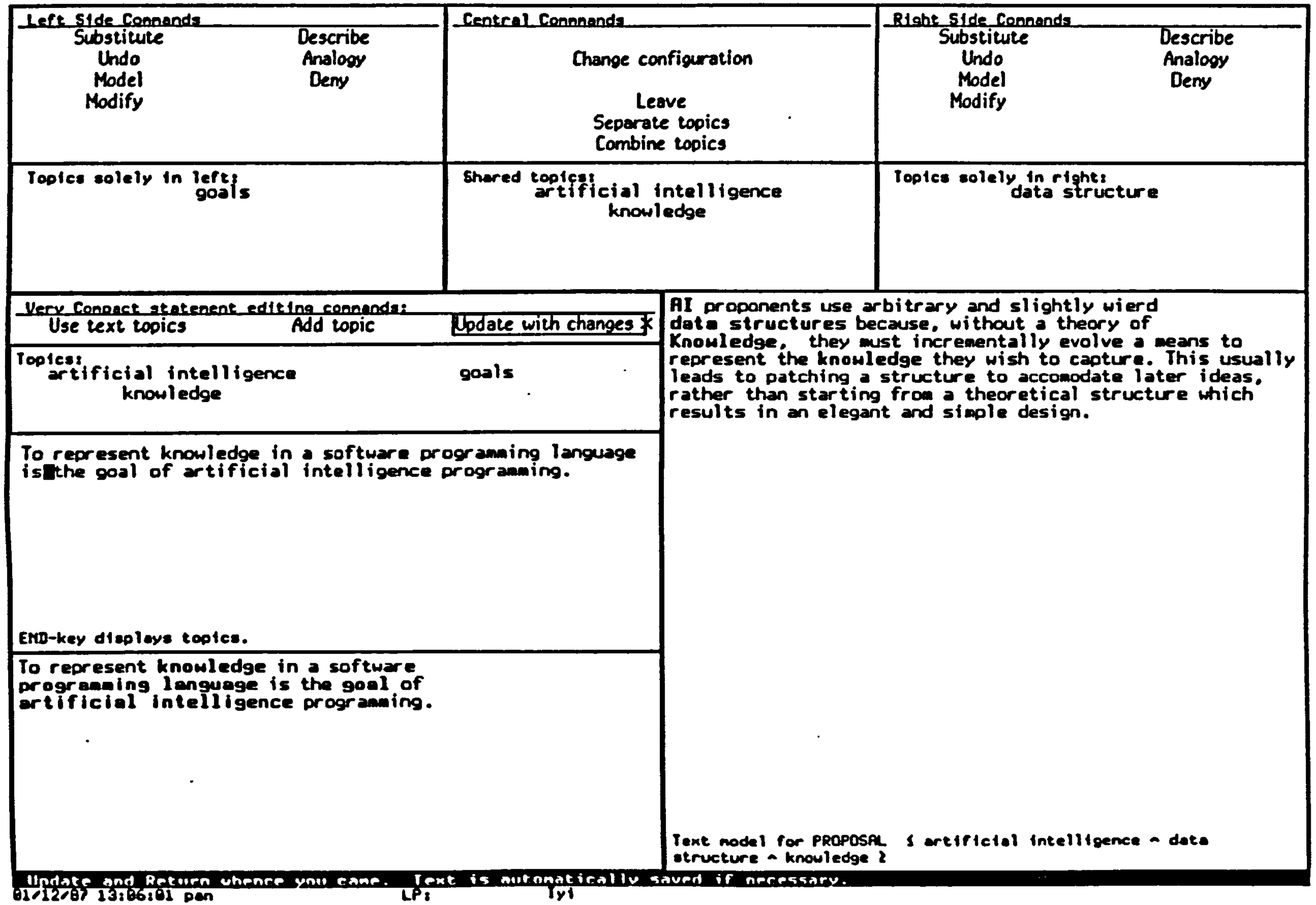

Left Side Commands
Substitute
Undo
Model
Modify
Describe
Analogy
Deny
Central Commands
Change configuration
Leave
Separate topics
Combine topics
Right Side Commands
Substitute
Undo
Model
Modify
Describe
Analogy
Deny
Topics solely in left:
goals
Shared topics:
artificial intelligence
knowledge
Topics solely in right:
data structure
Very Compact statement editing commands:
Use text topics
Add topic
Update with changes
Topics:
artificial intelligence
knowledge
goals
To represent knowledge in a software programming language is the goal of artificial intelligence programming.
END-key displays topics.
To represent knowledge in a software programming language is the goal of artificial intelligence programming.
AI proponents use arbitrary and slightly wierd data structures because, without a theory of Knowledge, they must incrementally evolve a means to represent the knowledge they wish to capture. This usually leads to patching a structure to accomodate later ideas, rather than starting from a theoretical structure which results in an elegant and simple design.
Text model for PROPOSAL { artificial intelligence ^ data structure ^ knowledge }
Update and Return whence you came. Text is automatically saved if necessary.
01/12/87 13:06:01 pan
LP:
Tyi

Figure 4

Left Side Commands

Substitute | Describe
Undo | Analogy
Model | Deny
Modify

Central Commands

Global Resolution
Change configuration
Leave | Separate topics
Combine topics

Right Side Commands

Substitute | Describe
Undo | Analogy
Model | Deny
Modify

Topics solely in left:
goals
programming language

Shared topics:
artificial intelligence
knowledge

Topics solely in right:
data structure

To represent knowledge in a software programming language is the goal of artificial intelligence programming.

Very Compact statement editing commands:
Use text topics | Add topic | Update with changes

Topics:
artificial intelligence | data structure
knowledge | LISP

AI proponents use arbitrary and slightly wierd data structures because, without a theory of Knowledge, they must incrementally evolve a means to represent the knowledge they wish to capture. These data structures are easily captured in a progrmming language such as LISP, which allows great flexibility in this regard.

END-key displays topics.

AI proponents use arbitrary and slightly wierd data structures because, without a theory of Knowledge, they must incrementally evolve a means to represent the knowledge they wish to capture. These data structures are easily captured in a progrmming language such as LISP, which allows great flexibility in this regard.

Text model for PROPOSAL { artificial intelligence ^ goals ^ knowledge ^ programming language }

[L,M] look for old topic first; [L] queries before making new one; [M] does not; [R] makes a new topic even if old topic
01/12/87 13:08:59 pan LP: tyi

Figure 5

| Left Side Commands | Central Commands | Right Side Commands |
|---|---|---|
| Substitute<br>Undo<br>Model<br>Modify<br>Describe<br>Analogy<br>Deny | Local Resolution<br>Change configuration<br>Leave<br>Combine topics<br>Separate topics | Attend to Self<br>Substitute<br>Undo<br>Model<br>Modify<br>Examine<br>Describe<br>Analogy<br>Deny |
| Topics solely in left:<br>goals<br>programming language | Shared topics:<br>artificial intelligence<br>knowledge | Topics solely in right:<br>data structure<br>LISP |

| | |
|---|---|
| To represent knowledge in a software programming language is the goal of artificial intelligence programming. | AI proponents use arbitrary and slightly wierd data structures because, without a theory of Knowledge, they must incrementally evolve a means to represent the knowledge they wish to capture. These data structures are easily captured in a programing language such as LISP, which allows great flexibility in this regard. |
| Text nodel for PROPOSAL { artificial intelligence ^ goals ^ knouledge ^ programming language } | Text nodel for PROPOSAL { artificial intelligence ^ data structure ^ knouledge ^ LISP } |

The proposed relations are coherent, but have conflicts elsewhere. Use this resolution and proceed to other conflicts.
01/12/87 13:09:54 pan LP: fyi

Figure 6

Ain Auditor 1: Cybernetics-and-all-that in context of Holist

Ain and Explore Commands:

| | | | |
|---|---|---|---|
| Environment | Add topic | Clear | What was |
| Explain | What next | Indicate understanding | Move text |

Topics:

knowables

Relations:

[ knowables • observer • order of universe ]
[ artificial intelligence • computer • knowables • knowledge representation ]
[ artificial intelligence • frames • knowables • Minsky ]
[ concurrent • Conversation Theory • knowables • Lp ]
[ coherent • knowables • relation • topics ]

PL calls this: knowledge
Holist calls this: knowables

It is used in the coherences:
[ artificial intelligence • computer • knowables • knowledge representation ]
[ Gordon Pask • knowables • Scientia ]
[ knowables • observer • order of universe ]
[ conversation • information transfer • knowables • participants ]
[ artificial intelligence • frames • knowables • Minsky ]
[ coherent • knowables • relation • topics ]
[ Conversation Theory • knowables • stable • universe ]
[ artificial intelligence • data structure • knowables ]
[ CASTE • Conversation Theory • knowables • knowledge representation ]
[ concurrent • Conversation Theory • knowables • Lp ]
**MORE**

AI proponents use arbitrary and slightly wierd data structures because, without a theory of Knowledge, they must incrementally evolve a means to represent the knowledge they wish to capture. This usually leads to patching a structure to accomodate later ideas, rather than starting from a theoretical structure which results in an elegant and simple design.

Text model for COHERENCE [ artificial intelligence • data structure • knowledge ]

01/12/87 13:19:22 pan LP: tyi

Figure 7

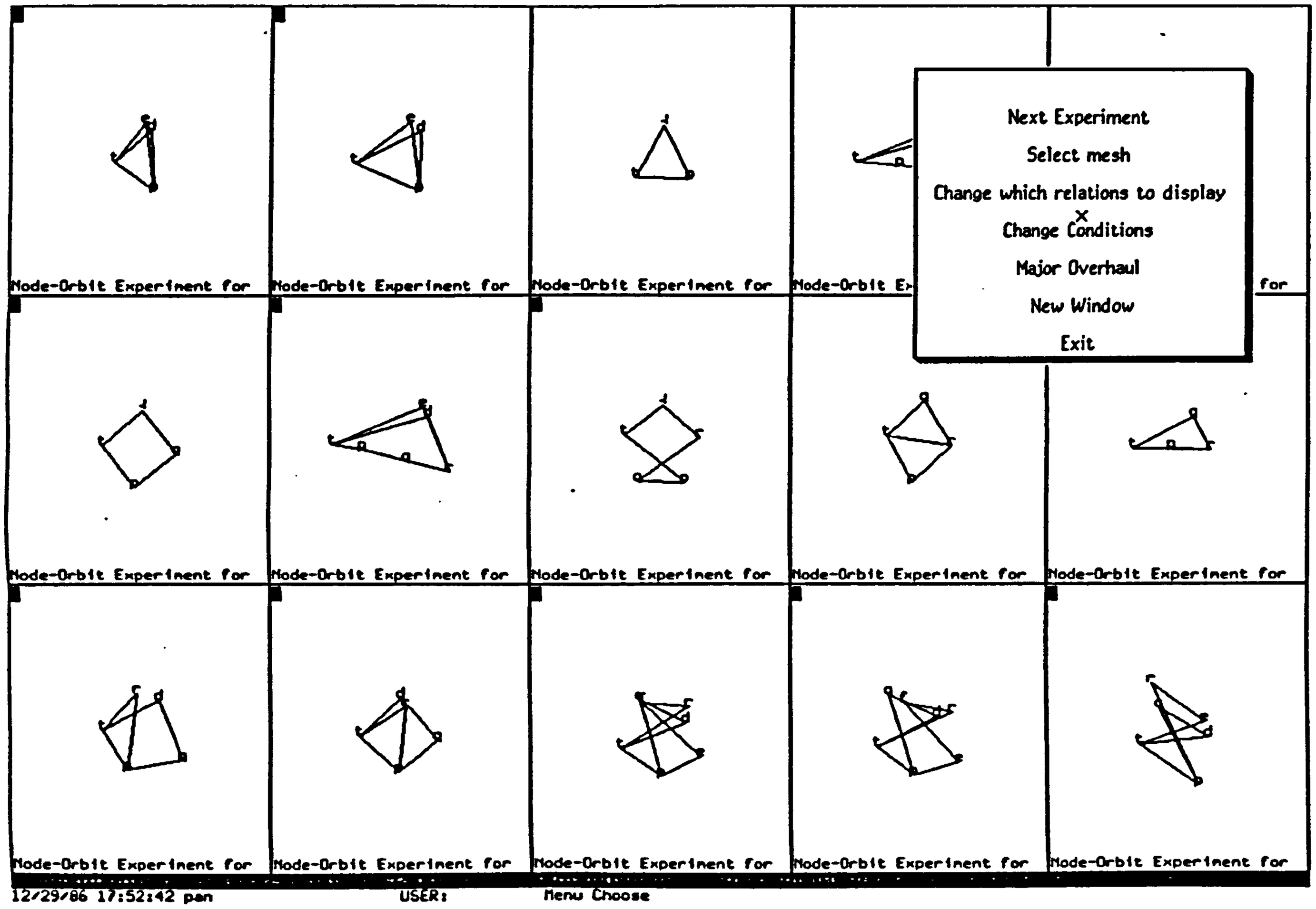

Next Experiment
Select mesh
Change which relations to display
Change Conditions
Major Overhaul
New Window
Exit
Mode-Orbit Experiment for
12/29/86 17:52:42 pan
USER:
Menu Choose

Figure 8a

Subset of Mesh to Display

*Do It*

[ t • p • q ]
[ t • p • r ]
[ t • p • q • d ]
[ t • p • r • e ]
[ ]
[ CASTE • ARI • Training ]
[ ARI • Army Research Institute • acronym ]
[ CASTE • entailment meshes • knowledge representation ]
[ Pask • conversation • Conversation Theory ]
[ ARI • Conversation Theory • decision support ]
[ PLATO • ARI • Training ]
[ PLATO • ARI • Training • CDC ]
[ CASTE • ARI • Training • Pask ]
[ PLATO • ARI • Off the shelf Training • CDC ]
[ CASTE • ARI • Developing Training Prototypes • Pask ]
[ ]
[ open door • key • clockwise ]
[ open door • key • anticlockwise ]
[ anticlockwise • clockwise • opposite directions ]
[ open front door • key • clockwise ]
[ open door • open back door • open front door ]
[ open back door • key • anticlockwise ]

Figure 8b

Node Orbit Experiments

*More above*

Rate: 0.2

Compute Distinction across entire mesh: Yes No
Compute Distinction across coherences: Yes No
R-law to use (1=linear 2=usual Newtonian): 2.0
Distinction strength: 1000.0

Compute Attraction within each coherence: Yes No
R-law to use (1=linear 2=usual Newtonian): 2.0
Attraction strength: 1000.0

Make trails of head node of Pruning: Yes No
Compute only the chosen head nodes: Yes No
How strongly: 1.0

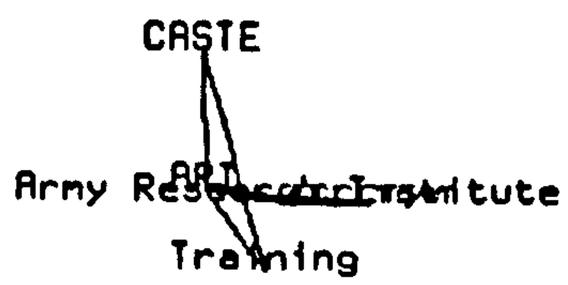
Compute Hierarchy within each coherence: Yes No
R-law to use (1=linear 2=usual Newtonian): 2.0
Hierarchy strength: 1000.0

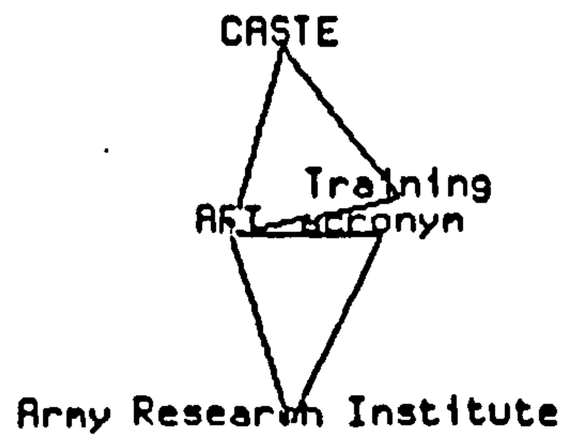
Use Random Initial Positions?: Yes No
How random an initial position (higher value for more spread): 30
How still before considered stopped (nil for never): none
Viscosity of ether (lower values are slower): 0.9
Compensation for long topic names (higher yields more spread): 0.5

*Bottom*

New Experiment ☐ Continue ☐

Figure 8c


CASTE
Training


CASTE
Training
Army Research Institute

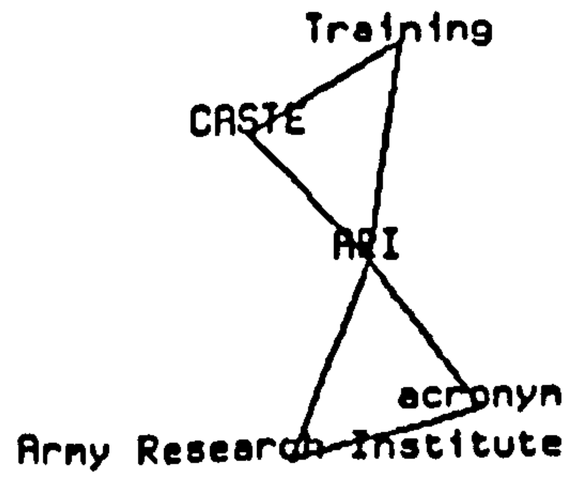

Training
CASTE
ARI
acronyn
Army Research Institute

Figure 9a - 9c

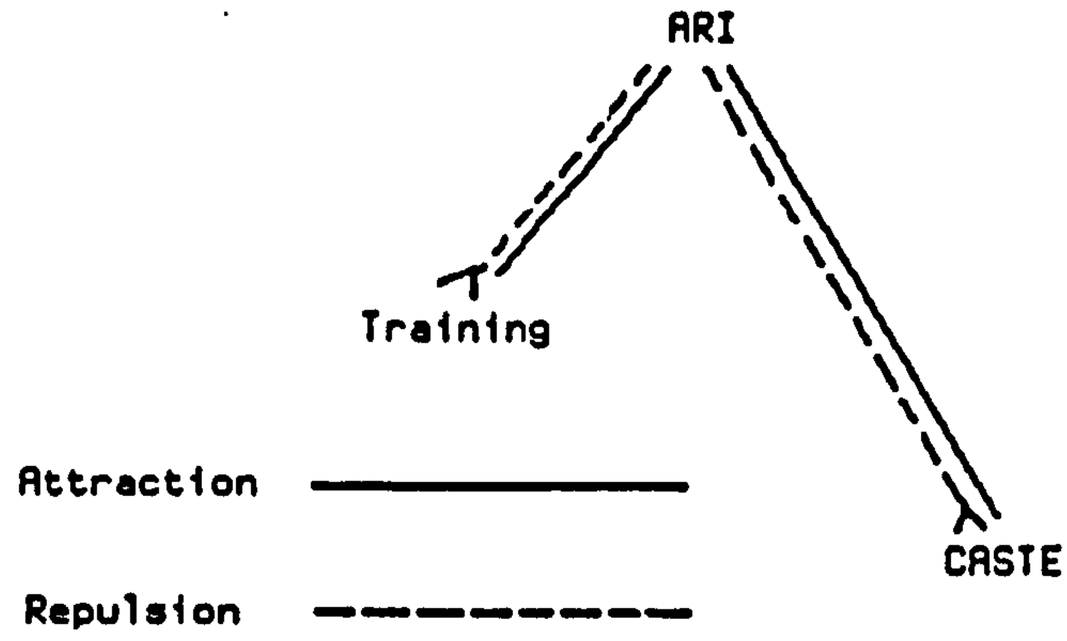



Figure 10

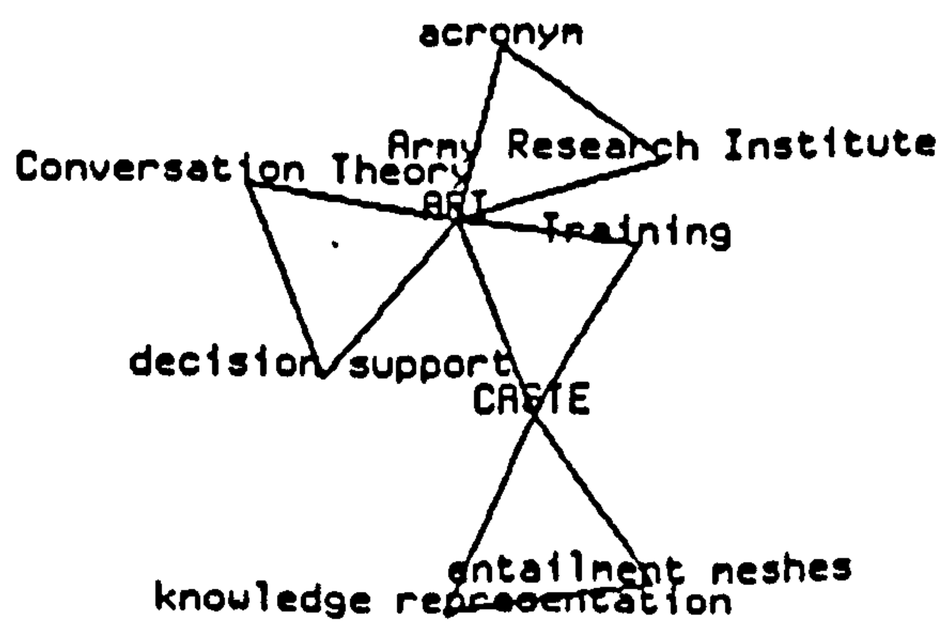
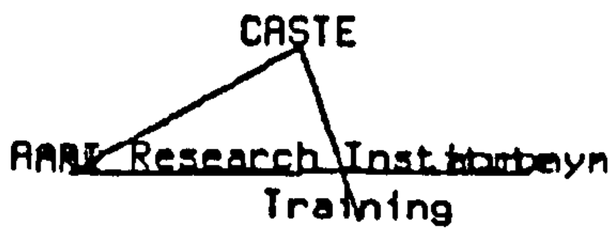
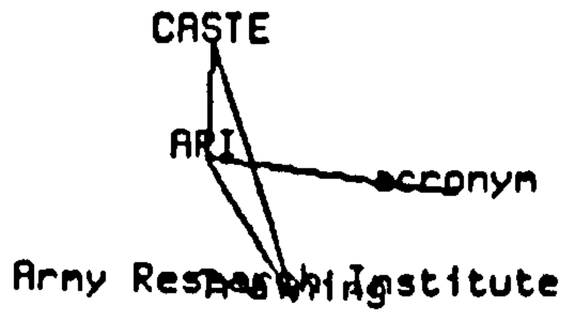
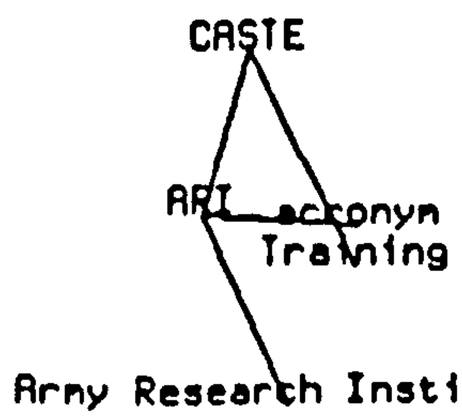



Figure 11

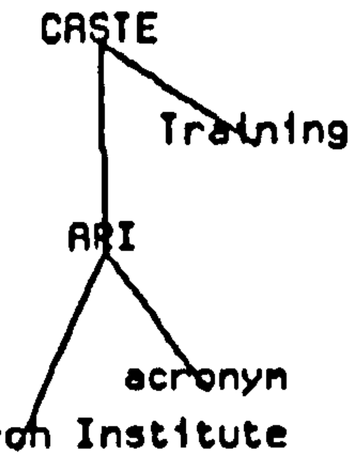
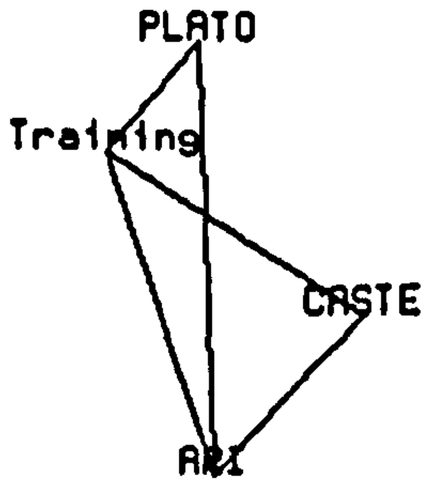
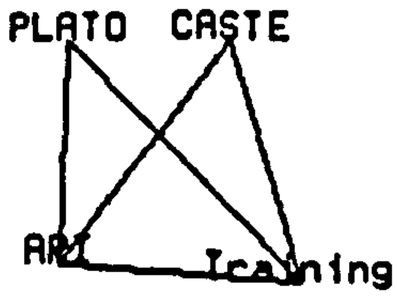




Figure 12a - 12e



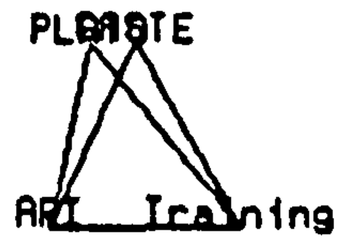
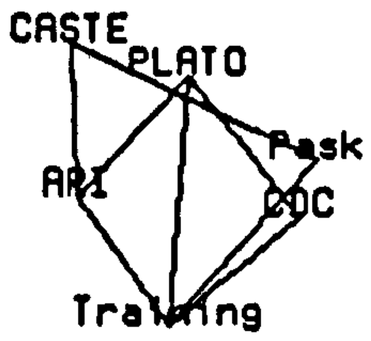





Figure 13a - 13d



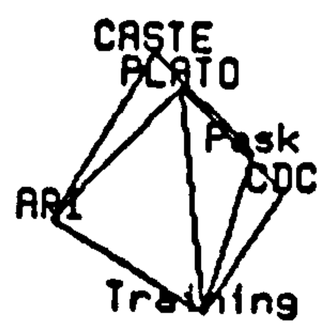





Figure 14a - 14c

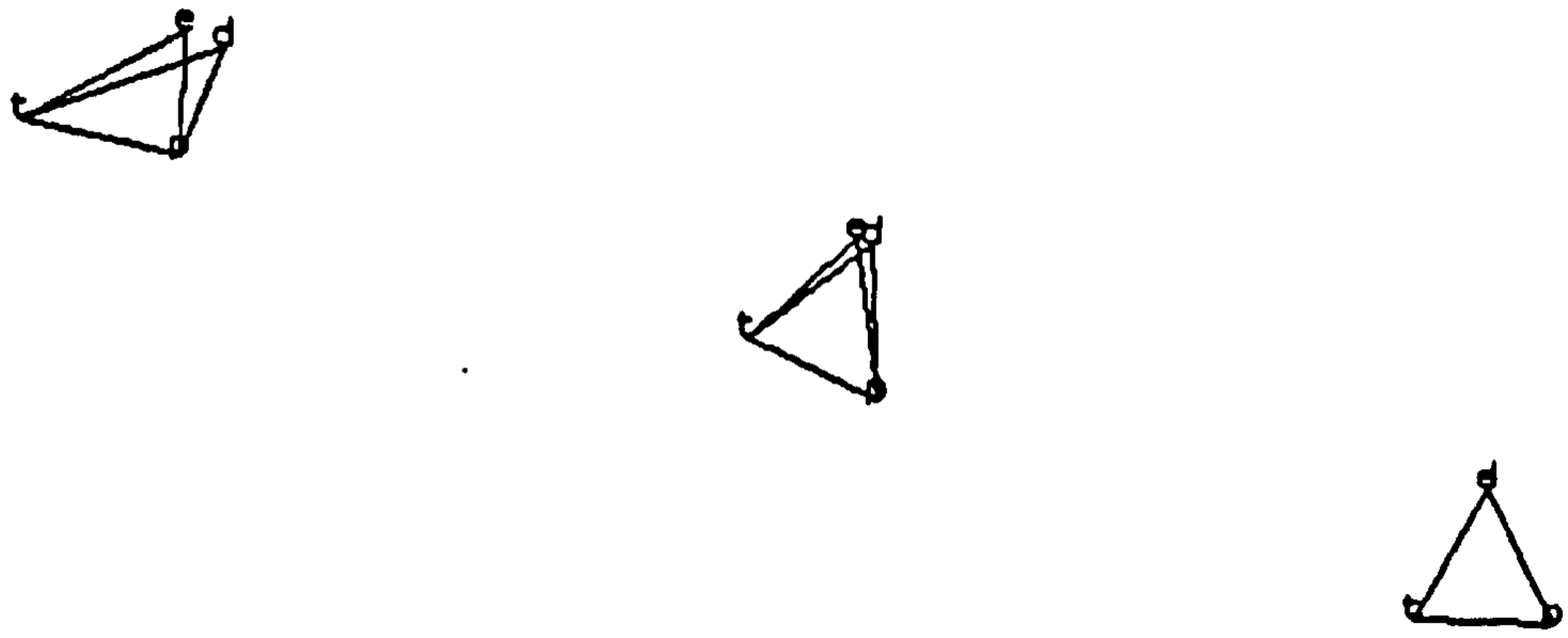


Figure 15 Case I

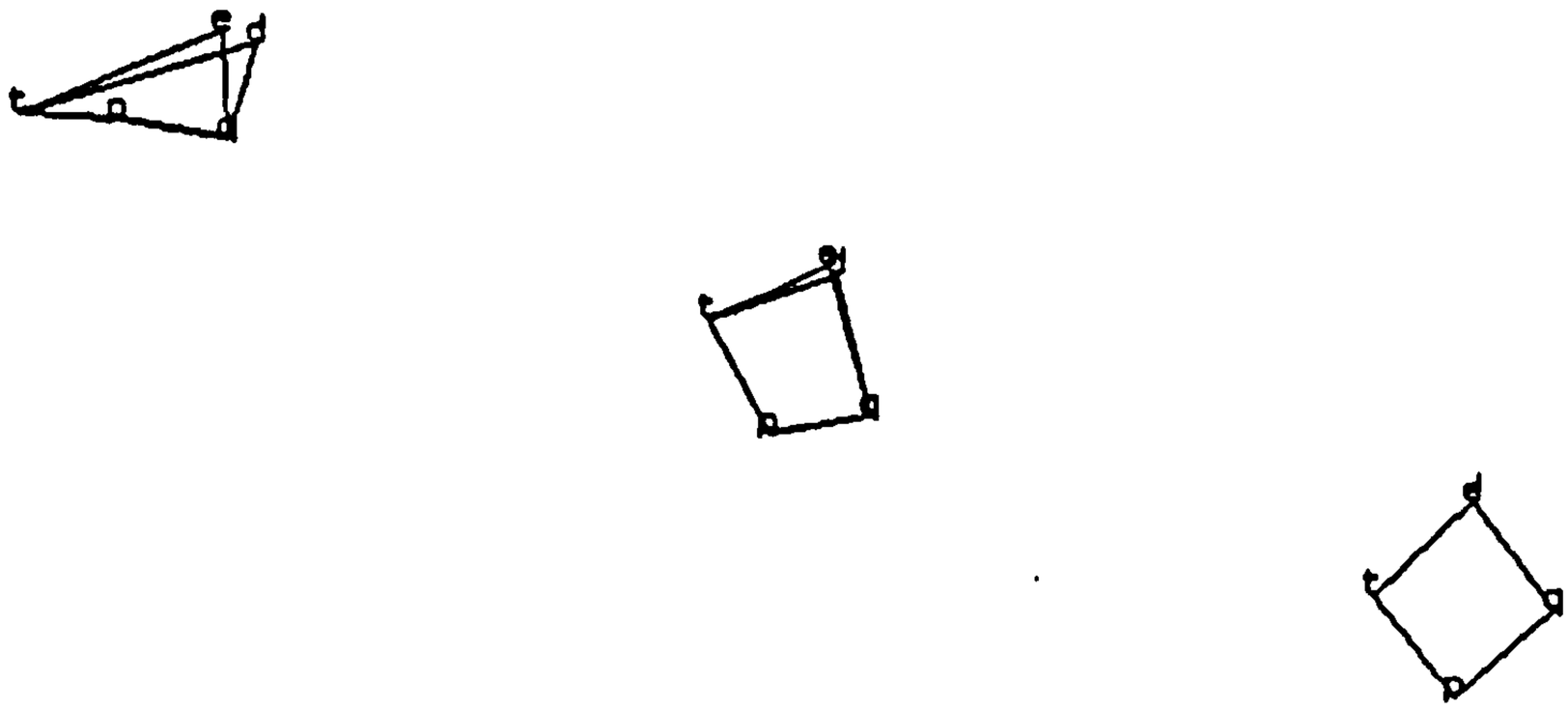
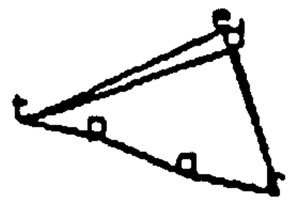


Figure 15 Case II



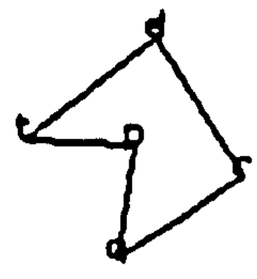
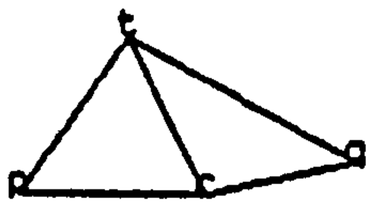


Figure 15 Case III



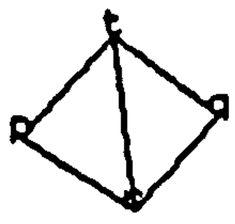
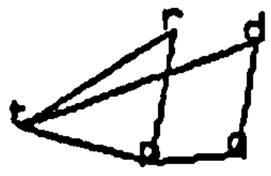
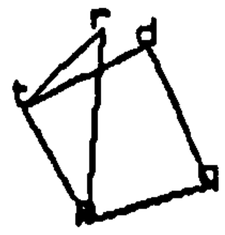


Figure 15 Case IV





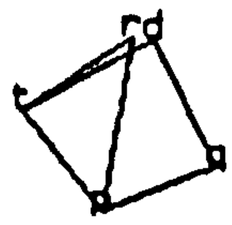
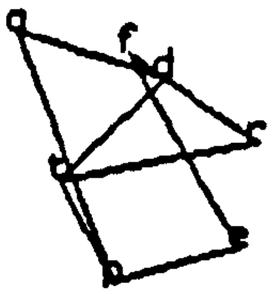
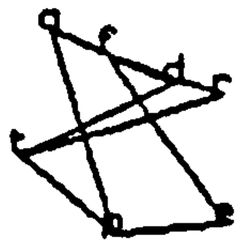


Figure 15 Case V





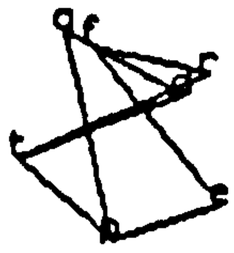
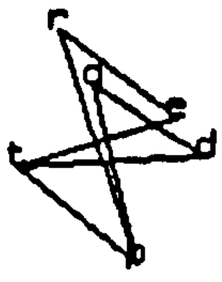
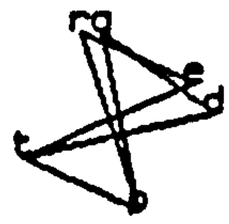


Figure 15 Case VI





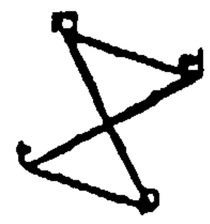




Figure 15 Case VII

ANNEX --- THOUGHTSTICKER User Manual

Attached to the Dissertation titled:

# An Examination and Confirmation of a Macro Theory of Conversations through A Realization of the Protologic Lp by Microscopic Simulation

A thesis submitted for the degree of Doctor of Philosophy

by

Paul A Pangaro

Department of Cybernetics, Brunel University

May 1987

# T H O U G H T S T I C K E R

Symbolics 3600 Research Implementation

# U S E R  D O C U M E N T A T I O N

JULY 1985

© PANGARO Incorporated
Washington DC
Richmond-upon-Thames UK

Table of Contents

# Table of Contents

# 1 Executive Summary: THOUGHTSTICKER

THOUGHTSTICKER is a software system for eliciting, representing and conveying knowledge. The system provides an environment for users to model their concepts and it detects conflict between the personal concepts of one user or between the concepts of several individuals. By comparing models of concepts that are in conflict, the system aids the user in reaching agreement between them.

The underlying knowledge representation scheme is a development of Conversation Theory (by Gordon Pask). As the title indicates, Conversation Theory asserts that thinking, learning, and decision making can be modelled as a dialogue, either internal to one person or between two or more individuals. THOUGHTSTICKER provides the means to model the concepts which are elicited and conveyed in that dialogue.

THOUGHTSTICKER is used for concept modelling in the processes of training, operational support, and strategic planning. Each of these processes involves a a dialogue, either between novice and expert or among several experts. The integration of these three processes is necessary since any complex human activity consists of all three roles: learning, problem-solving and forecasting.

THOUGHTSTICKER was developed under contract to the UK Ministry of Defense, and current applications include military training, decision making, and command and control. Other potential areas of application include office and industrial settings as well as traditional educational environments.

THOUGHTSTICKER shares many of the goals of expert systems such as capturing the knowledge of experts and offering provisions for expert advising. Unlike many expert systems, THOUGHTSTICKER does not require a knowledge engineer for the creation of knowledge structures. The system allows the expert to directly elicit and model his/her knowledge of any particular domain. Using the coherence logic of Conversation Theory to recognize conceptual conflict, the system also aids the expert in creating knowledge structures which are coherent. Similarly, THOUGHTSTICKER in the training mode has the capability for testing the student to determine if the student's knowledge is consistent with the expert's knowledge as modelled by the knowledge structure.

THOUGHTSTICKER utilizes the advanced window and mouse interface of the Symbolics 3600 LISP machine.

## 2 Preface: Symbolics 3600 Research Version

THOUGHTSTICKER is a software environment for the elicitation, representation and communication of information, based on the concepts of Conversation Theory. THOUGHTSTICKER contains a powerful knowledge representation system which plays an active role in the elicitation of knowledge from authors. The structure of the representation retains significant features of the knowledge such that a user, in the role of student, may use THOUGHTSTICKER as a learning environment without being restricted to a pre-defined or fixed learning strategy. In both authoring and tutoring modes, the system allows for "mixed-initiative" in that the system and/or the user may intervene in the interaction with the other.

Capabilities of the system include:

- Sophisticated window and menu facilities;
- Large capacity and high-performance;
- Fully developed functions for authoring and tutoring large subject matters;
- Unique tools for monitoring the internal consistency of the knowledge representation;
- Strategies for tutorial presentation and the ability for the user to vary the strategies;
- The ability to separate the contributions of different authors while monitoring the agreement and disagreement of content;
- Individualized vocabularies for each author as well as each user during tutoring.

The system which this manual documents is the first major LISP implementation of THOUGHTSTICKER. The system is intended as a research tool. The user interface presents all possible choices and requires that the user understand the basic implications of each choice.

The successor to this research version of THOUGHTSTICKER will be for users who are not familiar with its basis in Conversation Theory. This "Naive THOUGHTSTICKER" is being built on the present code and will contain a user interface which prompts the user on each transaction. So far as it is possible, the new version will keep its theoretical requirements hidden. The current, research version will continue to be incrementally improved and for some time will be the more powerful

version.

This user documentation has been prepared with these considerations in mind. Inevitably with a system undergoing constant improvement, certain details presented here will not correspond precisely to a particular release of the system. The user is encouraged to consult the various on-line HELP facilities within THOUGHTSTICKER, which will be updated as soon as features change and hence will be more accurate than this text. Additions and modifications to this manual will be provided periodically as Release Notes, and users are encouraged to respond with comments and questions to aid the process.

# 3 Introduction

## 3.1 Purpose of the Manual

The purpose of this documentation is to provide direction in the use of the THOUGHTSTICKER system in its current research form. It is geared toward a user who has had some exposure to the Symbolics 3600 interface (such as its mouse and keyboard) and the approach of Conversation Theory, although the basics of both are sketched below.

The documentation does not lead the user through each incremental step in the use of the system, pointing out each choice and the precise consequences of each action. Rather, the documentation notes the possible user options which are offered in the general case and remarks upon some of the situations in which those options might be taken.

## 3.2 Structure

The functions or modes of the THOUGHTSTICKER system are associated with frames of the screen and each major function is attached to its own frame. In general, the documentation will familiarize the user with how to use the functions of THOUGHTSTICKER and how to move about the system from frame to frame.

Each major section of this manual addresses a high-level aspect of the system in terms of a frame. In several cases, temporary frames exist in the system which are related to a particular primary frame and which offer a reduced set of options. These temporary frames are discussed in the major section about the frame they are related to.

Within each of these sections, there are two categories of explanation. The first is a walk-through of the function; the second is a glossary of commands present in the frame being explained.

## 4 THOUGHTSTICKER Vocabulary

This research version of THOUGHTSTICKER freely uses the terminology of Conversation Theory in its transactions with the user. In this section, the particular dialect of Conversation Theory used in THOUGHTSTICKER is described.

### 4.1 Entailment meshes: the knowledge representation

The knowledge held in THOUGHTSTICKER exists in an entailment mesh. The basic objects are the topic, coherence, analogy, ostension, model, contexture, and construct. There are also two temporary objects, termed suggestions and proposals. These objects may be grouped into several categories:

- The total database of THOUGHTSTICKER is referred to as an entailment mesh. The term is used to describe the rich interconnections of structure, reflecting the dependencies or "entailments" of meaning within the knowledge.

- The most primitive object in the entailment mesh is the topic. Topics are grouped by relations whose structure reflect the intention of the creator of the mesh. Coherences and analogies are types of relations. Ostensions are groups of topics which should be considered as single topics.

  These objects, although usually confined to a single mesh, may be present in several distinct meshes. Several entailment meshes may be examined in parallel in THOUGHTSTICKER by assigning different frames for each mesh.

- Proposals and suggestions are temporary forms of relations which may later be permanently added to the mesh as coherences, analogies, or (see immediately below) models.

- All of the above items may have one or more models. Models are not strictly part of the mesh; they are links from the mesh to an external world of communication or action.

- Contextures and constructs exist independent of any mesh; they represent the individual authors or users of the mesh and their conceptual styles and vocabularies.

Each of these is described in more detail in the following subsections.

## 4.2 Topics, coherences, analogies, ostensions

The basic unit of the THOUGHTSTICKER system is the topic. A topic is, almost without exception, named. The name of a topic can be any combination of characters of arbitrary length. To avoid confusion, topic names may not begin or end with punctuation marks.

A topic may have more than one name. For example, "AMTE," "Admiralty Marine Technology Establishment," "the Admiralty," and "Teddington" may all be names of the same topic. The name used by THOUGHTSTICKER in transactions with the user may be customized by the user, whether by the author during creation of the topic or by a user examining the mesh. Also, the same name may refer to more than one topic. When necessary, THOUGHTSTICKER queries the user to clarify any ambiguities.

A coherence is a relation between topics with the strong requirement that each topic in the relation can be "produced and reproduced" from the others in the relation. Thus, a coherent image or meaning for a Hungry Cat may be made from the topics Eat and Mice. But also, a coherent image of Mice may be made from Hungry Cat and Eat; and, finally, an image of Eat(ing) may be made from Hungry Cat and Mice. The strictness of the requirement allows for stable, redundant structures, resulting (minimally) in the ability for THOUGHTSTICKER to determine possible conflicts and ambiguities. If these conflicts and ambiguites are removed, the results are well-structured subject domains which are not self-conflicting.

The coherence is visually represented as a list of topics in alphabetical order, within square brackets, separated by a circled " + " sign: [ Hungry Cats ⊕ Eat ⊕ Mice ].

An analogy may be made between almost any of the THOUGHTSTICKER objects. The requirement for analogy is merely that some similarity and some difference be asserted between the objects. In the simplest case, the objects related by the analogy are topics. For example, there may be an analogy between Cheese and Mice, whereby both are Eaten By Cats but one is inanimate and the other is not. Note that an arbitrary number of differences may be asserted.

In the example, Cheese and Mice are called the "arms" of the analogy. An analogy may be between any number of arms. There is also a "diamond" of the analogy, in this case Eaten By Cats. The diamond represents what is intended as the similarity among the arms; it can be considered to represent the objects of the analogy at a "higher level." Collectively, the arms and diamond of the analogy are referred to as the elements of the analogy.

The visual representation within THOUGHTSTICKER consists of a list of elements within curly brackets, beginning with the diamond element, followed by a right arrow, "->" , followed by the arm elements

separated by double arrows, " <=> ". For example, the topic "Lisp" might be the diamond element of an analogy between "Maclisp" and "Interlisp"; this would be represented as:

{ Lisp -> Interlisp <=> Maclisp }

In some cases, there is no single THOUGHTSTICKER object (topic, coherence, analogy, etc.) which represents the element that a user wishes to use as the diamond or an arm of an analogy. If the desired element can be thought of as a loose combination of other already-existing elements, an ostension can be made. An ostension is visually represented by a list of elements within parentheses, separated by " + " signs. A statement of the form "Cats and dogs are similar to automobiles and homes in that too much money is spent on them" might be represented as:

{ Expensive -> ( Cats + Dogs ) <=> ( Automobiles + Homes ) }

where ( Cats + Dogs ) and ( Automobiles + Homes ) are ostensions representing the user's desire to group them for the purposes of the given analogy.

An ostension can be thought of as a transitory form. For example, in the first ostension (Cats + Dogs), it could replaced by an analogy representing the relation between "Cats" and "Dogs." This might be:

{ Pets -> Cats <=> Dogs }

The ostension could then be replaced in the previous analogy by the topic "Pets." The use of the ostension as a collective topic permits the user to defer analogy formation to a later occasion, or to omit the analogy entirely.

## 4.3 Temporary objects: suggestions and proposals

In the course of authoring, new relations may be formed which are not permanently incorporated into the mesh. Two different forms are used:

- During the process of saturation (Section 10, Saturate Seeker), the THOUGHTSTICKER system computes all legal relations that could be instated in the mesh as coherences. These are relations which are non-conflicting in the Conversation Theoretic sense. They are termed suggestions and are visually represented by a list of topics within inclusion brackets separated by logical "and" signs, for example:

C Dogs ^ Eat ^ Mice ⊃

- Prior to the instatement of any relation in the mesh, the relation is termed a proposal and are visually represented as a list of topics within pointed brackets, separated by logical "and" signs,

for example:

$$\leqslant \text{Dogs} \wedge \text{Eat} \wedge \text{Mice} \geqslant$$

## 4.4 Models

Each of the above abstract mesh objects (topics, ostensions, coherences, and analogies) may have one or more models. Models are concrete representations of the abstract object and refer to an external world or "modelling facility." In the present THOUGHTSTICKER system, models are restricted to text. This restriction is not fundamental; a model could be a still photograph, or a film, or a mechanical device. "Executing a model" could mean displaying the text to be read, showing the photograph, running the film, or activating the mechanical device. The current use of text to name topics and to represent models should not blur the distinction between them. The topic with the name "cat" could have as a model: "A cat is a gracile mammal with retractable claws." In the text of the model, the word "cat" does not represent a topic, nor do any of the other words represent any topic. In this way, the same model may be a model of many distinct topics. Coherences and analogies may also have models. For the analogy:

$$\{ \text{Lisp} \rightarrow \text{Interlisp} \iff \text{Maclisp} \}$$

one model might be:

> "Lisp is no longer a single computing language. The differing interests of workers in artificial intelligence and computer science at various corporations and universities have led to a divergence of Lisp dialects. Two of the most important versions are Maclisp, developed at MIT under the auspices of Project Mac, and Interlisp, developed at Stanford University in California."

For a model of a relation (coherence or analogy), one may specify an emphasis which characterizes the perspective (in the Conversation Theoretic sense) under which the model is a model of that relation. An emphasis consists of a list of topics or elements. Text models are visually represented by a portion of the beginning of the corresponding text and the names of the elements of the emphasis. For example, the coherence [ Hungry Cats $\oplus$ Eat $\oplus$ Mice ] is given an emphasis on the topic "Eat" and a text model which reads: "A constant characteristic of the relationship between cats and mice is that mice are eaten by hungry cats." The representation of the model would be:

"A constant characteristic...Emphasizing Eat."

## 4.5 Mesh Markings

In tutorial mode (described at length in Section 8, Aim Auditor), THOUGHTSTICKER maintains a model of the student's history. This is used to determine the precise sequence of presentation to the student during the teaching process.

THOUGHTSTICKER tracks what that user has and has not seen and "understood" up to that point in the interaction. The system maintains this history of interaction by a set of markings which are made upon the mesh, relative to a particular user. This subject is discussed immediately below and in greater detail in Section 14, Contextures and Constructs.

## 4.6 Contextures

The THOUGHTSTICKER system allows more than one individual to make additions to and use a mesh. Each individual is identified by a contexture, a term borrowed from Gottard Gunther. In Gunther's theory, a contexture is more than a context in which events are interpreted and evaluated, although the correspondence is close enough to allow the loose use of the latter for the former. Presently, a contexture within THOUGHTSTICKER is specified by a list of identifying names or texts, collectively referred to as the "author," and several lists of predicates.

These predicates are used, for example, to guide THOUGHTSTICKER in determining the sequence of tutorials. Consider the situation where two relations contain one topic in common. Either of those relations might be used to explain the topic, and THOUGHTSTICKER uses the predicates to choose between them. One such predicate might determine a clear preference for a relation which the trainee had not seen before, over one that s/he had seen. A tutorial strategy can be composed to conform to the student by choosing the predicates to apply and in what order. Currently, there are more than a dozen predicates used in the THOUGHTSTICKER tutorial mode. When creating a contexture, the user is led through a series of prompts to specify the predicates to be used for various purposes (see Section 14 on contextures for a more detailed discussion).

## 4.7 Constructs

The complete specification of an analogy includes similarities and differences that are embodied in the elements. The preliminary implementation of similarities and differences in THOUGHTSTICKER is through constructs, a term borrowed from the Personal Construct Theory of Kelly. A construct is a modifying attribute that can be associated with each element of the analogy. For example, in the analogy

§ Lisp -> Interlisp <=> Maclisp †, one suitable construct would be based on the distinction of dialect. Lisp is the general name of a computer programming language; Maclisp is used at sites influenced by Artificial Intelligence work at MIT; Interlisp is used at those sites influenced by Stanford University. The construct could consist of the adjectival phrases "Generic name," "Atlantic Coast dialect," and "Pacific Coast dialect." The constructs facility is described in more detail in Section 14 on Constructs.

## 5 Symbolics 3600 orientation

The THOUGHTSTICKER system is integrated into the Symbolics 3600 environment in a uniform manner in order to take advantage of the existing Symbolics software. In this section, a brief description of the relevant Symbolics features is given.

The standard THOUGHTSTICKER interface uses one or more windows on the screen. These windows may be organized into a permanent arrangement of several windows called a "frame"; [1] or, they may be "pop-up" windows that appear momentarily over the basic THOUGHTSTICKER frames. The latter allow the user to supply information or make a choice from a menu of options. Each of the primary THOUGHTSTICKER sub-systems is a frame of windows, which has specialized functions. The user communicates to these windows with the mouse and by typing at the keyboard.

### 5.1 On mice

Many of the words, symbols, and graphic illustrations on the screen are "mouse-sensitive." When the mouse cursor is brought near to a mouse-sensitive item, a box is drawn around the item to indicate that the object may be chosen with the mouse. Information about the item and/or the consequences of making such a selection is frequently supplied in the mouse documentation line at the bottom of the screen.

Since the mouse has three buttons, it is possible to vary the meaning of the selection by using different buttons. A click on the left mouse button may be denoted mouse-left or click-left. The convention is similar for the remaining buttons, middle and right. A double click on the right button would be denoted by click-right-twice and is reserved to retrieve the SYSTEM MENU of the Symbolics (the clicks must be in

---

1. The size and positions of the constituent windows can, in fact, be modified by any user at any time using the Edit Screen menu choice on the main SYSTEM MENU (reached by a mouse-click-right-twice, rapidly).

rapid succession). The use of the mouse for selection and control is described as "clicking," "choosing," or "selecting."

The Symbolics convention is that clicking on the left mouse button makes a simple choice, usually performing some default operation on the selected item, while click-right provides a further menu of possibilities. In THOUGHTSTICKER, a click-left may provide an abbreviated menu of choices, suitable for a relatively naive user, while click-right gives the more-experienced user access to a broader variety of options. Similarly, a click-left may be used to compress several operations into a single choice (such as offering a topic for possible tuition and providing an explanation of it), while a click-right permits the operations to be separated. Whenever there is a distinction between left, middle, or right clicks on the mouse, the mouse documentation line will explicate the choices.

In addition, the mouse is also used for "scrolling" the visible contents of windows whenever the size of the window limits the amount that can be seen at one time. When the mouse is moved to the left edge of the window, the shape of the mouse cursor changes. The mouse buttons are now commands for scrolling up or down through the contents of the window, changing the portion and position of the visible parts. Alternatively, single line scrolling is enabled by moving the mouse to the far right portion of the upper or lower edges of the window. The mouse documentation line will supply the necessary information in a complete but terse fashion in either scrolling mode. Note that the documentation appears only after the mouse has been moved to the left edge, in the one case, or the far right top or bottom, in the other case; a few experiments suffice to clarify its intent.

Many windows have "scroll bars" which indicate the fraction and relative position of the visible portion of the window contents with respect to the total contents; again, simple experimentation will clarify the function. Some scroll bars are permanently visible; some become visible only if the mouse is moved to the left edge of the window.

## 5.2 SELECT-ing Frames

The THOUGHTSTICKER system consists of a number of different frames which have specialized applications. The various frames are invoked by using the SELECT key on the keyboard. For example, the Aim Auditor is invoked by typing the single key marked SELECT and then typing the letter A; this is denoted SELECT-A. Although conventionally represented as an uppercase letter, the letter associated with SELECT-ing may be typed in lowercase. Use of the META key may also be necessary, denoted by "m"; for example, SELECT-m-G.

A complete list of all the THOUGHTSTICKER and Symbolics system programs that can be reached with the SELECT key may be examined at any time by typing SELECT-HELP on the keyboard.

### 5.2.1 Primary Frames

The primary THOUGHTSTICKER frames and their associated keys are:

- SELECT-T: THOUGHTSTICKER Central provides a large window for listing the topics and relations of the mesh, as well as performing certain overall bookkeeping operations.

- SELECT-A: The Aim Auditor is the most convenient frame for tuition and training. The user may choose topics to represent his/her "aims." THOUGHTSTICKER then aids the user in achieving understanding of these topics.

- SELECT-W: The Write Watcher is the principle frame for the authoring. A window with extensive word-processing features is provided.

- SELECT-S: The Saturate Seeker is an aid to authoring. Saturation is an Lp operation which determines what relationships between topics could be added to the mesh without conflict. These are presented as suggestions to the user, who may adopt, modify, or reject them.

- SELECT-m-G: (Experimental) Analogical Substitution of topics may be performed.

- SELECT-Z: (Experimental) The Zeta frame allows all functions of THOUGHTSTICKER to be reached from the same frame.

- SELECT-N: (Experimental) Beginner's frame, the "Naive THOUGHTSTICKER" interface.

With the exception of the experimental frames, detailed descriptions of the use of frames are given in succeeding sections.

There is no limit to the number of copies of each of these principle frames that may exist at the same time. One may, for example, have two Aim Auditor (tutorial) frames, which may be exploring different meshes or may be under the direction of different contextures.

It is important to note that by SELECT-ing another frame, a user has not foreclosed the possibility of returning to the original frame. SELECT-ing back will restore the user to the frame generally in the exact same state; exceptions include certain pop-up menus which may not reappear. The user typically does not exit a frame in the sense of terminating the activity of that frame; instead, s/he simply redirects his/her attention to an alternate task. The frame that is exited remains in THOUGHTSTICKER and awaits returning by a later SELECT.

Although the main THOUGHTSTICKER frames are initially designed to occupy the entire screen, the SPLIT SCREEN command of the main SYSTEM MENU can be used (as with any window) to place several frames on the screen simultaneously. In the current THOUGHTSTICKER system, it is practical to view and use two frames simultaneously, where the limitation is simply one of space.

### 5.2.2 Conflict Frames

There are two frames that deal with the discovery and resolution of conflicts. They are only entered when a conflict is encountered. If these windows are left without completing the process of resolution, they may be re-entered with the SELECT key. To symbolize the fact that the user can only re-enter them in this way, the META-key (abbreviated -m-) is added to their invocation.

- SELECT-m-C: The Conflict Clearance frame shows all of the previously instated relations with which a new relation is in conflict. Some simple conflict resolution can be performed.

- SELECT-m-R: The Resolver frame allows the most general resolution of conflict between two relations.

## 5.3 Menus and pop-up windows

There are a large variety of menus and other temporary windows used by THOUGHTSTICKER. Although they all are designed to allow the user to make a choice between options or to enter some new data into the system, they vary in the precise manner of their operation.

### 5.3.1 Text type-in

When the user is required to provide a small amount of text such as a name or short description, a temporary type-in window is provided. These windows appear when necessary and then disappear when the user has completed entering the needed text. The convention is that the text is completed with the END key; this permits RETURNs to form part of the text. The type-in window is properly activated (that is, capable of receiving typed input) if there is a blinking cursor in the window; if the cursor is not blinking, clicking the mouse in the window should restore it. The ABORT key is usually sufficient to cancel the request that prompted the appearance of a type-in window.

### 5.3.2 Command menus

Each of the THOUGHTSTICKER frames has a permanent command menu. These are usually general purpose commands which act on the frame and its contents as a whole. The mouse documentation line will contain

descriptive information about each command; this is accessed by moving the cursor over the particular choice, causing a box to appear around it. The action of these commands is typically final, that is, they cannot be directly undone or countermanded. In some cases, further commands are provided to restore an earlier state. Many commands lead to further choices (mediated, for example, by a pop-up menu). In these cases, the user may void the original choice by moving the cursor away from the menu without clicking on any choice.

### 5.3.3 Pop-up menus

Pop-up menus provide lists of choices, which appear temporarily on the screen. One use of pop-ups is to avoid placing infrequently used choices permanently on the frame. Pop-up menus are also useful to provide the user with commands that pertain to particular mouse-sensitive items on the screen. Thus, the user may click on a topic to obtain a menu of commands appropriate to that particular topic. These menus may be popped-up by pointing the mouse at one of the mouse-sensitive items and clicking. Clicking-left produces a simplified menu, while clicking-right produces one with all possible choices.

Pop-up menus will not disappear until a choice is made or until the mouse is moved decisively away from the menu area. Some of these menus may have explicit QUIT choices. If so, moving the mouse will not cause the menu to vanish.

### 5.3.4 Choice box menus

Some menus have "Choice boxes" placed in the bottom margin of the menus. These allow multiple choices to be made, or choices to be changed before the choice is acted on. Only by clicking on the "Exit" or "Return" box is the choice actually completed. These menus often have another choice box labelled "Abort" that cancels the request which prompted the menu; note that this is not always identical to making no choice and clicking on "Exit." There can be other, more interesting choice boxes which provide the user with additional options, and a few of the more common choice boxes are given at the end of this subsection.

For THOUGHTSTICKER menus with choice boxes, a different typeface is used to indicate the current status of the choice. Any chosen item will be redisplayed in boldface type to indicate the choice. By clicking on the item a second time, the user may "unchoose" the item.

These menus may permit only a single choice or, in certain cases, multiple choices. In the former case, the selection of one item will cause it to be redisplayed in bold and any previous choice will be cancelled. The choice is made final through the choice box labelled "Return bold choice" or "Return bold choices" as appropriate. If none

of the menu items is in bold, then the user has made the "empty" choice, which specifies that the user chooses none of the options. (For example, when specifying contextures, choosing none of the offered contextures will force the creation of a new one.) To undo the choice procedure entirely, the user should instead move the mouse away from the menu and perform a CONTROL-ABORT (see discussion below in Section 4.4 on Undoing and Unhanging).

When a menu with a choice menu first appears, one or more of the menu items may be displayed in boldface. This indicates, depending upon the circumstances, either a default choice or the current value.

The most common choice boxes used in THOUGHTSTICKER pop-up menus are:

- [Explore Bold]: The topics currently indicated in bold are explored, that is, tutorial mode is entered with the topics indicated (see Section 8 on the Aim Auditor for a discussion of Explorer frames).

- [Brief description]: Summary descriptions of the bold items are provided in a temporary window. For example, the description of a topic will include all its names, a list of the coherences in which it appears, and the constructs applied to it in any analogies.

- [Help]: Helpful information is provided in a temporary window. This expands on the terse documentation given in the mouse documentation line.

- [Return bold choice] or [Return bold choices]: These are used to complete the choice. The item or items in boldface type are the choice to be returned. If no items are in bold (an empty choice), THOUGHTSTICKER may initiate further inquiries depending on circumstances.

### 5.3.5 Choose-variable-values menus [sic]

The most complex menu used by THOUGHTSTICKER is called a "Choose variable values" menu by Symbolics. These menus are used to change or assign a value to some "variable." Each line of the menu refers to a different variable and the possible sorts of values that could be assigned to that variable. In some cases, each variable may use a different form of choice.

In one form of modifying the values of variables, a value for the variable is typed from the keyboard. To type in a value for a particular variable, the mouse is clicked on the value presently in

that line to select it.[2] The keyboard is used to enter the value and the RETURN key is used to terminate entry.

In the second form of modifying values, the mouse can choose from a list of possibilities that are presented; the current choice is displayed in boldface type.

Any choice can be altered again by selecting a second time. These menus are supplied with choice boxes at the bottom. If an "Abort" box is present, it will restore the values from before the menu was displayed. (For details, see the "Choice Facilities" documentation of the Symbolics.)

## 5.4 Help

In some circumstances, such as analogy formation, a HELP choice box or menu option appears on the pop-up menus. These will provide help specific to the choices being made by the user. Additionally, throughout the THOUGHTSTICKER system, the HELP key may be used to provide additional aid:

- In some circumstances, such as analogy formation, a HELP choice box or menu option appears on the pop-up menus. These will provide help specific to the choices being made by the user.

- The HELP key, used alone, provides information about the current THOUGHTSTICKER frame.

- Pressing HYPER-HELP provides more general information about THOUGHTSTICKER, which is applicable in all the frames.

- META-HELP pops up a menu of possible sorts of help from which the user may choose. Assistance is provided for particular commands, notation, and so forth.

- Due to a restriction within the Symbolics, there is one exception to using the HELP key alone. In the authoring frames, the HELP key is reserved by the Symbolics system itself for assistance with the many commands of the editing window. Thus, the SUPER-HELP key should be used to provide help with the frame as a whole; META- and HYPER-HELP are as above. The editing window HELP command prompts the user for a single character command, "Abort" (to quit), or "Help." For this last case, the HELP key is struck a second time and the various possibilities are explained.

---

2. If the current choice is empty, the Symbolics system code requires that the user move the cursor into the blank space following the named variable until a box appears, and then to click on the box to select the line.

Unfortunately, the current Symbolics system software has a bug which locks the editing window at this point. To get the explanations of the editing window help commands, the user must type FUNCTION-CONTROL-T.

# 6 Running THOUGHTSTICKER

## 6.1 Overview

The primary frames[3] of the THOUGHTSTICKER system are:

- The THOUGHTSTICKER Central Commands frame
- The Aim Auditor
- The Write Watcher
- The Saturate Seeker
- The Conflict Clearance frame
- The Resolver

There are also frames related to several of the primary frames; these related frames often have a reduced menu of options, are temporary, and are accessed through one of the primary frames. Each of these related frames is discussed in the section appropriate to the frame they are related to.

Every frame has a number of menu options. Menu options are either constantly visible in a Command menu, or are accessed in pop-up windows.

The primary and temporary frames have certain windows in common, described below. Although each of these windows may appear with differing headings and have characteristics which are distinct to the frame they are in, they share similar functions. Their frame-specific characteristics will be discussed in the section appropriate to each frame.

- Command menu: This window holds the basic menu options for each frame.

---

3. The Experimental frames are not included in the following discussion.

- Topic window: This window holds a list of current topics. In different frames, topic windows may be referred to as the "topic aimer," "topic list," or "topic window." If the list of topics is longer than the window, the scroll functions can be used.

- Relation window: This window holds a list of current relations. If the list of relations is longer than the window, the scroll functions can be used.

- Display window: This window is where the user views models for topics and relations. "Scrolling mouse sensitive text" initially labels this window along the bottom, indicating that display windows are sensitive to the mouse; the user may click words in these windows. Whenever text models are displayed in these windows, a label appears to identify the object being modelled.

- Editing window: This window offers the extensive word processing capabilities equivalent to the full ZMACS editor of the Symbolics.

Clicking on different objects each offers its own distinct menu options. Under certain circumstances, some options may not be offered. See Section 4.1, "On Mice," and Section 15, "System-wide Mouse Options."

## 6.2 Booting the System Without a Saved World

THOUGHTSTICKER is brought up by the sequence:

1. Enter to a LISP Listener via SELECT-L.

2. Type: (login 'tstik). You may select the newest or the released system.

3. Wait until the system comes up fully, which takes about 10 minutes. The login sequence will end with a "T".

4. Enter a THOUGHTSTICKER frame via the SELECT key, for example, SELECT-T. Note that when first booted, one version of each of the primary frames already exists.

5. Choose a stored mesh as per the options offered, or create a new mesh (see immediately below). The former choice is simpler to begin, as it allows immediate use of the system in its tutorial (Aim Auditor) mode.

## 6.3 Booting the System With a Saved World

1. If there is a saved world, there should also be a "boot file." Suppose for the following, that the boot filename is "tstik.boot".

2. At a lisp listener, logout if logged. Type: (logout).

3. Stop the machine. Type: (si:halt) or use the keys Hyper-Control-Local.

4. Boot the tstik world from the Fep> prompt. Type: <<boot >tstik.boot>>

5. Wait for the process to complete and then login. Type: (login :tstik :load-init-file) or (login 'accessname)

6. Go to step 4 and 5 above.

Note: Step 3 may take 40 minutes or longer if HUNKS or XT are loaded.

Note: the bootfile on Athena is correctly called xt-demo.boot.

## 6.4 Creating a New Mesh

The user can create a new mesh from any of the primary frames. The user clicks on [Environment] in the Command menu and then chooses [Create new mesh]. (see Section 6, Environment, for more detail.)

Until the new mesh is saved with [Store Mesh], it remains in memory but is not saved as a file.

Since this is a new mesh and there are no topics or relations, there are no useful and unique operations which can be performed in the THOUGHTSTICKER Central frame, the Aim Auditor, or the Saturate Seeker. The user should move to the Write Watcher frame in order to create topics and relations for the new mesh. Press SELECT-W and see Section 9, Write Watcher, for details on the use of this frame.

## 6.5 Moving from One Frame to Another

### 6.5.1 The SELECT Key

The user may move from one existing primary frame to another at any time by using the SELECT key. In general, most THOUGHTSTICKER frames can be accessed by pressing SELECT and the first letter in the name of the frame. For instance, the Aim Auditor is reached by pressing

SELECT-A; the Saturate Seeker is reached by pressing SELECT-S, etc. SELECT-HELP offers a list of THOUGHTSTICKER frames which can be accessed with the SELECT key.

### 6.5.2 The CONTROL Key

The user may create a new primary frame and move there by using the CONTROL key. For instance, the user presses SELECT-CONTROL-A, and creates a new Aim Auditor. After creating the new frame, the user must specify its mesh, version, and contexture.

### 6.5.3 The HYPER Key

The user may move from one frame to another and carry along the current mesh, contexture, and topic list by using the HYPER key. For example, the user can move from the Aim Auditor to the Write Watcher with SELECT-HYPER-W. The topic list from the Aim Auditor would be carried to the Write Watcher. This function is useful in many situations: when in the Write Watcher, the user may wish to move to the Aim Auditor in order to access the same topics in the tutorial mode; alternatively, when in the THOUGHTSTICKER Central Commands frame, the user may wish to move all of the topics of the mesh into the Saturate Seeker in order to saturate the entire mesh. Note that HYPER does not of itself create a new version of a frame; it merely carries information to an existing frame. Using both HYPER and CONTROL is possible; this will make a new version of a frame as well as carry information to it. For example, SELECT-HYPER-CONTROL-W.

### 6.5.4 The META Key

The META key used with the SELECT key will take the user back to a temporary frame. This only allows the user to return to a frame that has already been used. For instance, within the Write Watcher, the user can return to the last temporary Explorer window with SELECT-META-A. The Explorer is temporary and related to the Aim Auditor, and hence the command SELECT-META-A is mnemonic. Similarly, the user can return to a temporary editing window reached while in the Aim Auditor with SELECT-META-W. The temporary editing window is related to the Write Watcher, so SELECT-META and W is used. By pressing SELECT-HELP, the user can view a list of META options within the list of all possible SELECT commands.

## 6.6 Upon Entering Primary Frames

When entering each frame for the first time, the user is required to choose a mesh to instate by selecting the name of a mesh, a stored version of that mesh, and a contexture in which to work. (If no contexture is selected, a new one may be created at that point. See

Section 14, Contextures.)

## 7 The Environment Command

The [Environment] command appears on the command menu of each of the primary THOUGHTSTICKER frames. Choosing this command with the mouse will pop-up a further menu of commands which affect the general environment of the user. This section details the use of these commands.

### 7.1 Mesh commands

The first group of commands deal with the mesh that the frame is connected to:

1. [Describe mesh]: The user is given a short summary of information about the mesh. The number of topics, coherences, analogies and models is tallied.

2. [Create mesh]: A new mesh is created and the user is prompted for a name to identify the mesh. It is by this name that the mesh will be referred to in transactions with the user. The frame is redirected to the new (currently empty) mesh.

3. [Store mesh]: The mesh currently being used by the THOUGHTSTICKER frame is stored as a disk file. This allows the mesh to be retrieved at a later date.

4. [Restore mesh]: The mesh currently being used by the THOUGHTSTICKER frame is restored to the form contained in a previously stored disk file. If more than one version of the mesh has been stored, the user may choose which to use; the date and time of storing is displayed.

5. [Choose mesh]: This allows the user to redirect the window to another mesh. When this command is used, a menu appears which lists the meshes currently resident in THOUGHTSTICKER and the meshes which are stored as files on the system disk. The current mesh is displayed in boldface type. The choice is performed only after clicking on the box labelled "Return bold choice"; this allows the user to modify an initial choice. If a mesh that is already read from disk and resident in memory is chosen, the user is offered a further choice: simply switching to that mesh or restoring that mesh to the form it had when it was last stored on disk. If a mesh on file is chosen, the user is queried as to the version desired. If no mesh is chosen, the user is asked to confirm that a new mesh should be created. If he does so,

THOUGHTSTICKER proceeds as in [Create mesh].

## 7.2 Contexture commands

The next group of commands deals with the contexture through which the frame uses its current mesh.

1. [Describe contexture]: This lists the order of importance of the various predicates for the current contexture.

2. [Choose contexture]: A menu of the contextures available in the mesh is displayed. The current contexture is displayed in boldface. The choice is made with a choice box. A second choice box allows for the display of a brief description for the contexture that is shown in boldface. If no contexture is chosen, the user is led through a series of prompts to make a new contexture. The procedure is described in detail in Section 14 on Contextures.

3. [Edit Contexture]: The predicates used by the contexture for the various strategies (tutorial, saturation, etc.) may be respecified and their relative importance re-established.

4. [Edit Defaults]: This command relates to the ability of THOUGHTSTICKER to remember standard or "default" ways in which text is to be mapped into topics. Thus, the phrase "conversation theory" may, in differing situations, refer to the topics "conversation," "theory," or "conversation theory." THOUGHTSTICKER memorizes the particular situation (in terms of the precise text) and avoids asking the user to dis-ambiguate those situations in future. Defaults are usually set first in the Write Watcher frame, but they may be changed from any frame using this option of [Environment]. After clicking on [Edit Defaults], a series of pop-ups prompts allows the user to change the disposition of the defaults already set.

## 7.3 Constructs commands

The next group of commands deals with constructs, used in the formation of analogies:

1. [Describe constructs]: This displays a description of the constructs that are defined in the current contexture.

2. [Create construct]: Although constructs are typically created during the analogy process, the user may wish to create a construct by means of this command. The current constructs are described and the user is queried as to the number of new constructs. For each new construct, the user is led through a series of queries to create the construct; details are given in

Section 14 on Constructs.

### 7.4 Analogy commands

1. [Make analogy]: This permits the construction of an analogy between any of the mouse-sensitive mesh objects that are currently displayed. For example, one can make an analogy between several coherences. This is a limited form of the "condense" operation of Conversation Theory. Details are given in Section 13 on Analogy Creation.

## 8 SELECT-T: THOUGHTSTICKER Central Command Frame

### 8.1 Introduction

The THOUGHTSTICKER Central Commands frame is central to the THOUGHTSTICKER system because the user can list and examine the entire mesh as well as examine and alter the markings on the mesh. The markings on the mesh can be thought of as both an author's specification and a tutorial history. The markings include whether the objects have been accepted by a particular author and whether models have been seen by the learner and understood. Also from this central frame, the author may remove all topics from the mesh which are not currently in relations.

### 8.2 A Walk-through of the THOUGHTSTICKER Central Commands Frame

The user uses the command SELECT-T to enter this frame. When entered for the first time, the user is prompted to choose a mesh; if none is chosen, a new one is created. Alternatively, the user can create a new mesh by clicking on [Environment] and then [Create new mesh].

Given a mesh, the user clicks on [Operations] and then clicks on one of the options to list the items of the mesh: topics, relations, coherences, or analogies. For instance, the user clicks on [List topics] to see a complete list of the topics of the mesh in alphabetical order.

In order to aim for a topic in the mesh (i.e., learn about it), the user clicks on a topic in the list. The selected topic is moved to the Aim Auditor, which is the tutorial mode of THOUGHTSTICKER.

Alternatively, the user could click on [List relations] from the [Operations] command. If the user then clicks on a relation, the Aim Auditor appears, the topics of the selected relation will appear in the topic window, and the relations with those topics as members will

appear in the relation window (Section 8, Aim Auditor).

The most general command to view the contents of the mesh is [List anything]. A menu will appear with choices of all types of mesh objects; clicking any objects into boldface will cause them to be listed together after choosing "Exit." For example, clicking on "topics and "relations" will list all topics and all relations (coherences and analogies) present in the mesh. Any listed object may be moused, as before, entering the Aim Auditor. In addition, choices appear on the menu to modify the conditions under which the objects are displayed. For example, clicking on "topics" and "not understood" will list the topics whose marking (relative to the current contexture) is "not understood." The possibilities here are varied and are useful for both the author and student.

From this THOUGHTSTICKER Central Command frame, the user can move to any other primary frame by pressing its appropriate command: SELECT-A for the Aim Auditor, SELECT-W for the Write Watcher, and SELECT-S for the Saturate Seeker.

## 8.3 The Command Menu: "THOUGHTSTICKER Central"

Along with the THOUGHTSTICKER Central Commands window in this frame, there is a large display window with the name of the current mesh at its head. In this window the user can view the current mesh broken into its elements: topics, coherences, analogies, ostensions, and proposals. If the user clicks-left on a topic or a relation in the lists, that topic or relation will be carried to the Aim Auditor; this is one way of aiming for a topic or relation to be learned. This display window is distinct from all other primary frames in that clicking on topics and relations in this window does not offer mouse options.

The available commands are:

### 8.3.1 [Environment]

This option allows the user to change to a new mesh, to change the constructs, contextures, or to create an analogy. These options allow the user to reshape the conditions of the environment of the current user transactions. See Section 6, Environment, for further details.

### 8.3.2 [Operations]

This option is unique to this frame. The user clicks here to examine the mesh and to alter its markings. There are two kinds of options within [Operations]: The user can view how the mesh is organized in terms of topics, relations, coherences, or analogies; or, the sub-choice [List anything] offers a multiple-choice menu for picking

## 9.1 Introduction

This frame is the tutorial mode of THOUGHTSTICKER. It allows the user to "aim" at topics and to view the models for those aims and the relations (coherences and analogies) which contain them. This aiming process is a tutorial exploration. Each user has an individual tutorial history, which is a result of both that user's actual movement within the system and the contexture which defines preferences for the strategy of presenting new material.

The student is free to aim at will and the actions of the student are monitored and a tutorial history is recorded. Based upon these tutorial markings, the current contexture determines the particular model that will be offered to the student when the student aims for a topic; and the new topic and/or new relation that will be offered when the student asks for new material.

The Aim Auditor frame consists of several windows: a command menu ("Aim and Explore Commands"), a topic window ("Topics aimed for"), a relation window ("Relations"), and two display windows in which to view the text models. Each of these windows is customary except for the following: the topic window lists current topic aims; and the relation window lists relations which include those tutorial aims. The command menu window is discussed below.

## 9.2 A Walk-through of the Aim Auditor Frame

The Aim Auditor is entered when an object is moused from the THOUGHTSTICKER Central frame; alternatively, this frame allows the user to type in a topic to learn with the [Aim for new topic] command. The user may then view a model for that topic with the [What is] command. If the user wishes to view a model for a relation that has that topic as a member, the [Why is] command is used. The user can wipe the tutorial screen clean with [Clear], or make room for a second model with [Move text].

If the user is viewing a model for one relation, and spots a relation in the relation window or a topic in the topic window which s/he wishes to explore, the user can click on the object and choose options from a pop-up window. See Section 15, "System-wide Mouse Options," for a full discussion of the options offered when clicking on objects of the mesh.

If the user wishes to look at the entire mesh, SELECT-T will move to the THOUGHTSTICKER Central Commands frame and the options are listed by [Operations]. Then, returning to the Aim Auditor can be accomplished by clicking on any object listed.

The user may continue aiming for topics and viewing text models until a topic is understood, that is, until the user is confident of understanding it. [Indicate understanding] allows the user to mark topics as understood.[4] This marking will affect which topics are suggested by the system when the [What is], [Why is] and [What Next] commands are used.

## 9.3 Command menu: Aim and Explore Commands

### 9.3.1 [Environment]

This shows the options that the user has in terms of the organization of the mesh. The user can view the statistics for the elements of the mesh: topics, coherences, analogies, and models. The user can also explore and alter the current contexture, explore and create constructs, and create analogies. For more detail, see Section 5, Environment.

### 9.3.2 [Add new topic]

This option allows the user to add a new topic aim. The user types a topic name in the pop-up window provided and then presses END to complete the transaction. The new topic appears in the topic aimer. If new topics that are typed are similar to other topics in the mesh (e.g., "relate" and "relations"), a pop-up window shows which topics the new aim corresponds to and asks the user to choose the intended topic. If none of the corresponding topics are intended, the user may exit the window without any choices. If no topic can be found, THOUGHTSTICKER offers to make a topic of that name.

### 9.3.3 [What is]

This option is used to view a topic model for one of the aim topics. The user clicks on [What is] and THOUGHTSTICKER displays a model for one of the topics in the topic aimer. On each succeeding click, the system moves through each of the topics and displays their models until all available models of the aim topics have been shown.

---

4. This is a passive form of the "teachback" of Conversation Theory but is possible within the present THOUGHTSTICKER only as such a domain-free feature. Note that manually marking the topics "understood" is not strictly necessary to receive a sensible tutorial sequence; it does offer an additional level of subtly to the process, however.

### 9.3.4 [Why is]

This option is used to view a model of one of the relations. THOUGHTSTICKER chooses the "best" relation to show a model of, depending upon the user's tutorial history and the current contexture (Section 14). On each succeeding click, the system chooses a new "best" relation based on the history, including the model just seen. This continues for each click until models for all relations have been shown.

### 9.3.5 [Explain]

This option combines the [What is] and [Why is] commands into a single sequence, passing automatically through the direct models of the aim topics to models of their relations. The original [What is] and [Why is] actions are preserved on the [M] and [R] mouse buttons. In some frames [What is] and [Why is] are omitted for reasons of space.

### 9.3.6 [Clear]

This option allows for clearing of specific parts of the frame. Click-left clears the topic window; click-right pops-up a menu of clear options: topic window, relation window, text display windows, clear all. This function is uniform throughout THOUGHTSTICKER.

### 9.3.7 [What next]

Depending upon the user's tutorial history and the current contexture, THOUGHTSTICKER suggests a new aim topic which "best" suits the user. The topic window is cleared, the suggested topic added, and the relation window is refreshed with a list of relations which include the new topic aim. Click-left on this option also shows a model of the "best" relation that appears in the display window, and hence is the usual means of moving on to the next topic to be learned; click-right merely displays the new topic, useful for quicker movement through the mesh.

### 9.3.8 [What was]

This option takes the user to topic and relation lists from the previous transaction. By clicking on [What was], the user can move back sequentially through the frames of previous transactions. [What next] moves the user forward again, but if additional explorations have been made, the same path may not be repeated due to changes in the tutorial history.

### 9.3.9 [Move text]

This option moves the text model from the main display window to the secondary one. The user may now view another text model in the primary window. This feature is useful for comparing text models.

### 9.3.10 [Indicate understanding]

By clicking left on this option, the user receives a list of the topics in the current topic window and in the most recently viewed relation. The user can mark the list indicating which topics are understood and which are not. By clicking middle on this option, the user can indicate understanding for all topics which are in the relation window as well as the topic window.

## 9.4 The Explorer: A Related Frame

This temporary window is accessed in several ways. One way is by clicking left on a topic or relation and choosing the pop-up window option [Explore]. Another way is to choose Explore in one of the temporary windows. The Explorer is related to the Aim Auditor frame in that it has many of the same command windows and command menu options. The command menu options in the Explorer which are also in the Aim Auditor are:

- [Move text],
- [Explain],
- [What is],
- [Why is],
- [What next], and
- [What was].

For explanation of these options see previous section.

There is also a mechanism for returning to the primary frame from which the Explorer was reached. The following command menu option is only offered in the Explorer and not offered in the Aim Auditor:

### 9.4.1 [Return]

This option takes the user from the temporary Explore frame back to the previous primary frame used.

## 10 SELECT-W: The Write Watcher and Related Frames

### 10.1 Introduction

The primary functions of this frame allow the user to build relations between topics and to compose text models.

One approach to the Write Watcher is to create topics, and then to create a relation by naming the topics which comprise it. The relation may be instated as a coherence or an analogy. The user has the option of creating a text model for that relation. If the user has not created a text model in the current editing window, a message will appear after the relation is instated, "This relation has no models."

On the other hand, the user may create a text model, build a relation, and then assign the model to the relation. THOUGHTSTICKER can discern topics which appear in the text of the model, which the user may then make deletions or additions to. Topics which are assigned to a relation model need not appear in the model as words or phrases.

The user may also create a text model and attach it to one or more topics (see [Create model] in Section 15, "System-wide Mouse Options"). A model for a topic serves as an explanation for that topic without necessarily involving other topics.[5]

Technically, models (whether for topics or relations) are not instated into the mesh. Topics and relations are instated; the models are accessed by means of the objects they model.

There are four windows in the Write Watcher frame: The command menu ("Statement editing commands"); a topic window ("Topics necessary to this relation (conflicts are shown)"); an editing window; and a display window. These windows are customary except: The topics in the topic window indicate which topics have been assigned to the relation currently in development; the topic window displays any pre-existing,

---

5. A coherence, in contrast, requires the assertion of a number of topics, each of which may be explained in terms of all the others; see Section 3, THOUGHTSTICKER vocabulary.

conflicting relations in the mesh as the user builds the proposed relation;[6] text models are typed into the editing window and then appear in the primary display window after the user presses END to indicate the end of the text model; the last text composed remains in the secondary display window until END is pressed again. The command menu options are discussed below.

## 10.2 A Walk-through of the Write Watcher frame

The usual strategy for creating models and building relations is to create a text model and, in parallel, build a relation by assigning topics to the topic window. When the topics in the topic window are instated as a relation, the current text model becomes a model of the relation.

### 10.2.1 Creating the text model

1. The Write Watcher is always prepared to receive text from the keyboard.[7] The text model is typed in, using the word processing commands of the Symbolics.

2. At any time, pressing END will display in the display window the current state of the text, with identifiable topics in bold. The assigned topics will appear in the topic window. THOUGHTSTICKER may interrupt with queries about how to dis-ambiguate topics and word phrases when it cannot do so automatically. The user can set these defaults permanently into the mesh with the [Make Default] choice box on the pop-up menus which query the user.

---

6. This has the effect that the topic window also displays relations as they are instated in the mesh. The interpretation in this case is that if the topics in the topic window were instated <u>again</u>, they would conflict with the last instated relation.

7. The unfortunate exception to this generalization occurs if the operating system interrupts with messages; for example, announcements about garbage collection. In this case, the cursor in the editing window will not be blinking; clicking-left over the window with the mouse will revive the blinking of the cursor and the window is ready for input. For resolving other difficulties, see Section 4.4.

### 10.2.2 Assigning topics to the Relation

At this point, the user has created a text model and needs to build a relation which the text will model. This is done by indicating the topics that are to be associated with the relation. There are several means to do this:

- Topics may be assigned to text models by clicking on words in the text model in the display window. When the user clicks left on a word, a pop-up window prompts the user to make the topic into a topic word or phrase. Clicking on [Make this word a topic] immediately makes that word into a topic and it appears in the topic window. Alternatively, clicking on [Build a topic word phrase] initiates a mode where each click-left on a word adds it to a growing phrase which is to become the topic. For example, "Once and future king" would require four separate clicks to build the phrase. After the last word of the phrase is clicked on, clicking middle or pressing END will end the phrase. The phrase then becomes a topic and appears in the topic window. At any point, pressing RESUME cancels the current phrase being built. During this transaction, the phrase in development will appear in the lower corner of the display window.

- Topics are also assigned to the relation by using the option [Use text topics] which retrieves as topics all words and phrases in the current text which are topics in the mesh. After the model is typed in and END is pressed, words which are already-existing topics will appear in boldface type in the display window; using [Use text topics] will move these topics to the topic window. Again, THOUGHTSTICKER may query about certain cases and [Make Default] should be used in these pop-ups to store the proper mapping of text to topics.

- Topics are also assigned by using the option [Add topic] which allows the user to type in a topic that is not a word or phrase in the text model. After clicking on [Add topic], the topic word or phrase is typed into the pop-up window, and terminated with the END key. A detailed description of the [Add topic] command is given in section 9.4.4.

### 10.2.3 A Conflict Before Instantiation

In the course of authoring as just described, the relation under development is not yet present in the database. If, at any stage in assigning topics the tentative relation has a conflict with any existing coherences, the existing coherences will appear in the topic window. The user may alter the text and topics for the current relation in order to avoid conflict. The user may:

- Add topics to the relation with [Add topic], or,

- Remove topics from the relation by clicking on topics in the topic window and then choosing [Remove from window], or,

- Resolve the (potential) conflict before instating the relation; the user clicks right on the existing relation in the topic window, chooses [Resolve this conflict] in the pop-up window, and moves directly to the Resolver (Section 12).

### 10.2.4 Instating the Relation

After the text model is complete and the relation is built with the desired topics assigned to it, the relation may be instated into the mesh as an analogy or coherence. The user clicks on [Instate]. The current text model will be attached to this relation. (The user may also decide to assign the current text as a model to any or all topics in the topic window.)

There are several cases which will occur when the user attempts to [Instate] a relation:

- The relation has too few topics to be a valid coherence.[8] In this case a pop-up window notifies the user of this fact and provides the options to [Make this an analogy] or [Make this a model]. See Section 13, "Analogy Creation," and Sections 9.2.6 and 9.2.7.

- The relation is legal. In this case, the user receives a message to this affect in a pop-up window with the option to choose whether this text model should be instated as a model for a coherence, an analogy, or a topic. The options are: [Make this a coherence], [Make this an analogy], and [Make this a model]. See the sections following.

- The relation conflicts with existing coherences. The existing coherence(s) appear(s) in the topic window. See the section "A Conflicting Relation," below.

---

8. As noted, one requirement for coherence is that any topic in the relation may be "produced and reproduced" from its neighbors in the relation. It would therefore seem possible to have a coherence with just two topics, whereby each "produced" the other. This, however, is in contradiction to the further requirement for coherence: each topic must maintain its distinction from other topics in the coherence. Without distinction, any topic might become ambiguous with another (unable to be distinguished), leading to loss of structure within the mesh. Two topics do not of themselves contain a distinction between each other; hence, they need a third for a production/reproduction that is stable.

### 10.2.5 Making the Relation into a Coherence

The user chooses [Make this a coherence]. A pop-up window prompts the user to choose the topic(s) which express the emphasis for this model. The topic or topics chosen are marked for this model, showing the user's intention or focus in creating the model. Providing an emphasis for coherences will affect which relations are viewed in the Aim Auditor, but an emphasis need not be given.

If the relation is accepted into THOUGHTSTICKER, it will appear in the topic window along with the topics of that relation. This indicates that a second instantiation of this same relation would encounter a conflict.

After instating the legal relation, the user is free to use any number of options in the Write Watcher frame or to move to other frames. See Section 5, Running THOUGHTSTICKER, for options on how to move to different frames. To begin another text model, return to the Section 9.2.1 above, Creating the Text Model.

### 10.2.6 Making the Relation into an Analogy

After clicking on [Instate], the user may make the topics in the topic window into an analogy. Click on [Make this an analogy]. A series of pop-up windows ask the user to provide the elements of the analogy. The topics of the relation are used as the basis from which to choose these elements, although other topics or ostensions may be added. See Section 13, "Analogy Creation."

### 10.2.7 Making the Relation into a Model

When the user clicks on [Instate], the text model may be entered as a model for a topic or topics. In a pop-up window, the user clicks on [Make this a model]. Another pop-up window asks the user to choose which topic or topics are modelled by the current text. The current text becomes a topic model for the chosen topics.

## 10.3 A Conflicting Relation

If the proposed relation conflicts with existing relations, there are several steps which can be taken:

- The user can mark the existing relation as not accepted. The user clicks right on the existing relation in the topic window and chooses the pop-up window option
  [Mark relation as invalid (not accepted)]. Then the user may [Instate] the proposed relation again.

- The user can merge the proposed relation with the existing relation. The user clicks right on the existing relation in the topic window and then chooses
[Merge new relation with this relation] in the pop-up window to merge the two relations. The proposed relation is forgotten and the current text model becomes another text model for the existing relation.

- The user can resolve the conflict. The relation(s) in the mesh that conflict with the proposed relation will appear in the topic window and a pop-up window will indicate that the proposed relation cannot be entered as a coherence. This pop-up window will also give the user the options of trying to resolve the conflict; of making the relation into an analogy; or of making the text model into a model for a topic (the latter two both avoid the conflict issue). The menu options are:

  * [Try to resolve],

  * [Make this an analogy], and

  * [Make this a model].

For details on the analogy option, see Section 13, "Analogy Creation." If the user chooses the model option, the system will prompt the user to choose which of the topics in the proposed relation will receive the current text model as a topic model. If the user chooses to resolve the conflict, the Conflict Clearance frame appears; see Section 11, "The Conflict Clearance Frame."

The user can exit the pop-up window just discussed without making a choice between coherence, analogy, or model. This returns to the Write Watcher frame in order to resolve the conflict by modifying the proposed relation. The user can add more topics to the proposed relation in the editing window of the Write Watcher with the [Add topic] option in order to give more distinction to the relation. Alternatively, the user may remove topics from the proposed model by clicking right on topics in the topic window and then clicking on [Remove from window]. After changing the topics for the text model, the user presses END to see if there are still conflicts. If there are, the conflicting relation(s) will appear in the topic window.

The user may also resolve the conflict between an existing relation and a proposed relation by modifying <u>both</u> relations. The user clicks on the existing relation in the relation window of the Write Watcher or the Conflict Clearance frame, then clicks on [Resolve this conflict] and enters the Resolver frame directly. See Section 12, "The Resolver Frame."

## 10.4 Command menu: Statement Editing Commands

### 10.4.1 [Environment]

[Environment] shows the options that the user has in terms of the organization of the mesh. The user can view the statistics for the elements of the mesh: topics, coherences, analogies, and models. The user can also explore and alter the current contexture, explore and create constructs, and create analogies. For more detail, see Section 6, Environment.

### 10.4.2 [Use text topics]

Click on this option to assign to the relation in development, the words in the current text model which are already topics in the mesh. These words can be viewed in boldface type in the display window by pressing END at any point in the composition of the text model. A pop-up window and message may appear which asks the user to choose which topics in a list of topics are appropriate to the coherence. The user merely clicks on the topics and exits the window.

### 10.4.3 [Clear]

This option allows for clearing of specific parts of the frame. Click-left clears the topic window. Click-right pops-up a menu of clear options: topic window, relation window, text display windows, clear all. If the user has created text in the editing window and has not saved it, a pop-up window will prompt the user to choose whether to save the text or not.

### 10.4.4 [Add topic]

This option permits the introduction of new topics. A pop-up window will prompt the user for the name of the topic, which is terminated by the END key. The mouse buttons control the detailed action:

- [L]: If no topic with that name is found, the user is queried to determine if a new topic with that name should be made.

- [M]: If no topic with that name is found, a new topic is automatically made.

- [L, M]: If a topic with the exact name is found, it is used. If topics with similar names are found, the user is queried through a pop-up window to determine which if any of the similarly named topics are appropriate.The user may:

* explore any topics listed in bold,

* obtain brief descriptions of the topics, or,

* make the new name the default name for some topic to avoid future queries.

- [R]: Makes a topic with the indicated name even if one already exists.

### 10.4.5 [Retrieve lost work]

There are three categories of object relating to "lost work": text, models, and proposals. Text is created whenever the user types into an editing window, and all text which is created in editing windows is saved to the file system. Models (consisting of text sentences) are created and attached to a topic or a relation; if the user kills the topic or relation, the model will remain unattached to the mesh but is still available in the file system. Proposals (consisting of lists of topics) are created when a relation is built which may or may not have a text model.

[Retrieve lost work] allows the user to access all models and proposals which are not currently being used in the mesh, or text which is not assigned as a model. As usual, a reminder of the meaning of mouse clicks, explained in depth below, is synopsized in the mouse documentation line when the mouse is moved over the [Retrieve lost work] option and the cursor box appears.

- Clicking left on [Retrieve lost work] will display a list of all text which has been stored in the file system and which is not being used in the mesh. The file will have an arbitrary name given by THOUGHTSTICKER, relating to the contexture that created it. If the user has not left the Write Watcher since the last time this option was chosen, a pop-up window will prompt the user to either [Get last list (faster)] or [Go to disk (accurate)]. The former merely repeats the last list it displayed, which may be superceeded by user actions since then; the latter re-reads the disk file status but is somewhat slower.

- Clicking middle on [Retrieve lost work] will display any models which were created but which do not have a relation or topic attached to them. If there are any, a portion of their text will appear in a list in a pop-up window and the user may choose one. This causes the model to be displayed in the editing window. At this point, the user is free to continue by building a relation for the model or assigning it to a topic.

- Clicking right will display a list of proposals that were created (for instance, in the Write Watcher or a related frame) but not instated in the mesh. If the user chooses one of the proposals,

its elements and any conflicting relations will appear in the topic window of the Write Watcher frame. If a single text model exists, it will appear in the editing window. If there is more than one text model for the proposed relation, a pop-up window will prompt the user to choose the appropriate one. At this point, the user may continue in this frame and alter the text model for the proposal if desired, or instate the proposal with its text model as is.

Once an unattached text, model, or proposal is displayed in the Write Watcher, the user may expunge it from the files with [Kill if not used].

### 10.4.6 [Instate]

Click on this option to instate the topics in the topic window as a relation into the mesh. If there are no conflicts, a pop-up window prompts the user to make the relation into a coherence, an analogy, or a model. Once the user chooses coherence or analogy by clicking on the option, the coherence or analogy is instated in the mesh with the current text model in the editing window attached to it. The new relation appears in the topic window. If the user chooses to make a model rather than a relation, a pop-up window prompts the user to choose which topics in the topic window shall have the current text attached as a model.

### 10.4.7 [Kill if not used]

Once the user has used the option [Retrieve lost work], the user may remove that text from the file system if it is unwanted. Choosing this option will remove the topics of the current relation from the topic window. The text for the killed relation in the editing window can be removed with [Clear]. A pop-up window will prompt the user whether or not to save the current text.

### 10.4.8 [Pop up earlier state]

Click on this option to bring back an earlier text.

## 10.5 Related Frames: Temporary Editing Windows

Under certain circumstances, the user may be given access to a temporary editing window. For example, in The Resolver, the user has the option to [Modify] a text model; in this case, a frame which is related to the Write Watcher appears. These related frames are temporary, offer a reduced menu of options, and are accessed from other primary frames. Refer to the appropriate sections above for details on specific functions.

# 11 SELECT-S: The Saturate Seeker Frame

## 11.1 Introduction

This frame is used as an aid to authoring; it produces suggestions of possible legal relations. The user specifies the list of topics to choose from, and the number of topics to be included in each suggestion (the "adicity of the relation"). For example, an adicity of 4 means that 4 topics are to be included in each suggestion. The Saturate Seeker generates a list of suggestions, any one of which can be legally instated. Once one of the suggested relations is instated, the mesh changes, and the suggested list of legal relations will also change. The process of saturation is divided into groups of topics to produce suggestions from, each of which is called a saturation "world."

Saturate Seeker has three main windows: the command menu ("Saturation Commands"), a topic window ("Topics being saturated"), and a relation window ("Suggested relations"). Each of these windows is customary except: the options popped-up when clicking on suggestions are relevant only to saturation. The command menu is discussed below.

## 11.2 A Walk-through of the Saturate Seeker

Typically, a user will manually add topics to the topic list, expand the list to include topics which are related to this list of topics, and then begin the saturation process. If the user would like THOUGHTSTICKER to determine a group of topics to saturate, the user may click on 'Suggest Topics.' THOUGHTSTICKER uses the contexture's saturation predicates to pick the topics most in need of saturation.

The saturation process is often a long one, and the user can "examine" the suggestions before the entire process is complete. Once a list of current suggested relations is examined, the user typically will remove topics or relations from the list and examine the saturation world again. This is a process of narrowing down the suggestions to more meaningful relations.

When one of the suggestions is appealing, the user clicks on it and is automatically moved to a temporary editing frame. There, the user builds a relation, creates a text model, and instates the relation. The following walk-through is one example of such a saturation process, and the user is encouraged to explore the full potential of this frame by referring to the command menu outline which follows this section.

1. If the topic list is not clear, the user clicks-left on [Clear].

2. The user clicks on [Add new topic]. A pop-up window prompts the user for a topic name to be typed in, after which the user presses END. The topic name will appear in the topic window. The user may continue adding topics to the list in this manner.

3. If the user would like to include all topics which are in relations with the topics in the topic window, the user clicks on [Expand]. Each topic that shares a relation with the original topic list will appear on the list.

4. To remove a single topic from the topic list, click left on the topics and then click on [Remove from window]. To clear the entire topic list, click left on [Clear].

5. The current list of topics should now consist of all those topics to be used in producing suggestions. The user clicks on [Start up saturation]. THOUGHTSTICKER prompts for the number of topics to be included in each suggestions (the "adicity") and any topics which are to be included in every suggestion. The process of saturation then begins in the background. The user may begin saturating any number of saturation worlds; for each one, the user is prompted for topics to include in every suggestion, focusing the process. There is no visible sign of when the saturation process is at work or complete for each world.

6. To view the results of saturation, the user clicks on [Examine]. Because a number of worlds may be saturated simultaneously, a window lists the saturation worlds and the user chooses which world to examine. Clicking-left on a world will display the current suggestions. The list of saturation worlds shows the adicity and required topics of each. For economy, each saturation world pauses when it has completed 20 suggestions. Any action by the user which eliminates any suggestions, or requests that the process carry on, causes the system to compute more suggestions.

7. After viewing the relation list, the user may want to remove certain relations from it. The user clicks on a relation and a pop-up window will prompt with the following possibilities:

   - If the user chooses [Discard this suggestion], a pop-up window will display the topics of the suggestion and prompt the user to choose one or more of them. Further suggestions which contain the topic or topics in combination will be discarded. This is a way of narrowing the scope of the saturation. After all relations containing the topics are discarded, the suggestion relation list will be refreshed.

   - If the user chooses [Adopt this suggestion], a temporary editing window titled "Compact statement editing commands" will appear. This temporary window is related to the Write Watcher frame. In this case, the user will create a

relation model for the selected relation from the Saturate Seeker. This window has the options: [Use text topics], [Add new topic], [Instate], [Pop up earlier state], and [Return]. The topics of the selected relation appear in the topic window of this temporary editing window. The user types in the text model for a relation and adds or takes away topics from the topic window as desired. Finally, the user confirms the topics assigned to the current relation by pressing END and chooses to [Instate] the relation and its text model (see Section 9, Write Watcher). The user clicks on [Return] to return to the Saturate Seeker. The suggested relation list is now updated; any suggestions which are rendered illegal by the instantiation of the coherence are automatically removed from the relation window.

8. The user may examine another saturation world with [Examine], or continue to manipulate the current suggestions list. If the user would like more suggestions, clicking-right on a world and choosing [More] will replace the list with more suggestions.

9. If the user clicks left on a topic in one of the suggested relations, the topic can be explored, described, etc., or removed from the window. A pop-up window offers several options; see Section 15, "System-wide Mouse Options."

## 11.3 Command menu: "Saturation Commands"

### 11.3.1 [Environment]

[Environment] shows the options that the user has in terms of the organization of the mesh. The user can view the statistics for the elements of the mesh: topics, coherences, analogies, and models. The user can also explore and alter the current contexture, explore and create constructs, and create analogies. For more detail, see Section 6, Environment.

### 11.3.2 [Pop up earlier topics]

This option clears the current topic list and replaces it with the previous topic list.

### 11.3.3 [Suggest Topics]

THOUGHTSTICKER provides a list of topics to saturate.

### 11.3.4 [Add new topic]

This option adds a topic to the topic window as in the Write Watcher. A pop-up window prompts the user to type the topic name; END completes the transaction.

### 11.3.5 [Clear]

This option allows for clearing of specific parts of the frame. Click-left clears the topic window. Click-right pops-up a menu of clear options: topic window, relation window, text display windows, clear all.

### 11.3.6 [Expand]

This option will enlarge the current topic list. For each topic in the current topic list, each relation is examined and all topics of each relation are added to the topic list.

### 11.3.7 [Examine]

This option allows the user to view the results of saturation. A window lists each of the saturation worlds initiated by the user. Click-left on a world lists the current suggestions. Click-right pops up a menu of the following options:

- [Show]: List the current suggestions.
- [More]: Add to the list of suggestions by computing more and displaying them.
- [Drop]: Clear the current list of suggestions and refresh it with a list of newly-computed suggestions.
- [Kill]: Abort the calculation of suggestions for the world and eliminate the world.

### 11.3.8 [Start up saturation]

This option begins the saturation of a specific set of topics selected by the user. This set and its saturation is a "saturation world." The following steps take the user through a saturation process:

1. A pop-up window prompts the user to supply the adicity of the suggested relations; the adicity denotes the maximum number of topics to be included in the suggested relations. The user

clicks on the number in the window, supplies a new number, and presses RETURN. There must be three topics to a coherence (see footnote in Section 9.2.4); therefore, the minimum adicity for a saturation world is 3.

2. Once the adicity is provided, the topic window prompts the user to indicate which topics are required to be in each of the suggested relations. The topics included in each suggestion by this means constitute a "theme" with which to focus the saturation process. The user need not require that any topics appear in the suggestions; returning without clicking on any topics accomplishes this.

3. The saturation process begins in the background. The user may saturate any number of saturation worlds at one time since they work in parallel. There is no visible sign that the saturation process is working or complete for each world. See [Examine] above for details on how to access the saturation worlds.

4. If, during the saturation process, the user attempts to list suggestions of a saturation world for which there are more possible suggestions, the message appears: "There are more suggestions that could be explored given more time."

## 11.4 The Topic Window: "Topics being saturated"

This window shows the current list of topics which will be used in the suggested relations list. It can be cleared with the menu choice [Clear] or it can return to an earlier state with the menu choice [Pop up earlier topics]. This list is referred to as the topic list or the topic window.

## 11.5 The Relation Window: "Suggested relations"

In this window, the user views the suggested relations for a specific saturation world.

## 12 Conflict in the Mesh

The strict requirement for coherence is that each topic in the relation may be produced and reproduced from the others in the same coherence. The result of this is that certain types of contradiction and ambiguity may be detected mechanically by THOUGHTSTICKER.

For example, suppose that this coherence was instated in the mesh: [ Sky ⊕ Color ⊕ Blue ]. The implication is that, if asked to combine the topics Sky and Color, the topic Blue would be reproduced. Now suppose that a new coherence was proposed: [ Sky ⊕ Color ⊕ Red ]. Of itself, this new coherence would not present any problem. If both were to co-exist in the same mesh, consider the answer to the question, "What do Sky and Color, when combined, reproduce?" One coherence implies Blue and the other implies Red. Clearly, both are not identical and the mesh contains contradictory information.

Changing the mesh to eliminate the conflict is a process called "resolution." There are a number of approaches to the elimination of conflict, and in some circumstances any one of them may be applied:

- Merge the two relations because they actually intend the same meaning but use different names for the same topics.

- Split, or "bifurcate," one or more of the topics which overlap both relations. Thus, Sky could become Sky at Noon in one case and Sky at Dusk in the other.

- Add further topics to the relations. Adding the separate topics Noon to the first coherence and Dusk to the other would eliminate the conflict.

Each of these actions are sufficient to resolve the conflict because they eliminate any uncertainty in the reproduction of topics, given a set to combine. In the last resolution, the combination of Sky and Color would produce nothing; Sky and Color and Dusk would un-ambiguously reproduce Red. (A full discussion of the meaning behind resolution of conflict in meshes, the variety of possible outcomes, and the role of analogical forms in bifurcation is beyond the purview of this manual.)

There are two primary frames within THOUGHTSTICKER which provide the user with the opportunity to resolve conflicts in the mesh: The Conflict Clearance frame and The Resolver. The Conflict Clearance frame provides access to each existing conflicting coherence, and it allows the user to make mechanical changes to the coherences in question by

changing the mesh status of those relations (e.g., accepted, not accepted). Whereas this frame handles the status of the relations, The Resolver frame allows the user to alter their content. The Resolver allows the user to work directly with the elements and meaning of the relation, rather than just change the status of the relation in the mesh.

## 12.1 The Conflict Clearance Frame

This frame is accessed when a conflict arises between one (or more) existing coherences in the mesh and a proposed coherence. A Conflict Clearance frame cannot be initiated by the user; in the event of a conflict upon instating a relation, this frame is always displayed by the system. SELECT-META-C is used to return to a previously used Conflict Clearance frame.

The purpose of this frame is to serve as an intermediate stage between the system's notification of one (or more) conflicts and the resolution of those conflicts. The user may examine the conflicting coherences before choosing one with which to resolve the conflict. Choosing a coherence takes the user from the Conflict Clearance frame to The Resolver. If after resolving a conflict with one existing relation in The Resolver, more conflicts still exist in the mesh, the user is returned to the Conflict Clearance frame.

This frame consists of a command menu ("Conflict Overview Commands"), two relation windows ("Proposed relation" and "Conflicting relations"), and two display windows. Each of these windows is as before, with the following distinctions:

- The proposed relation appears in the "Proposed relation" window while a list of existing, conflicting relations appears in the "Conflicting relations" window;

- The text model for the proposed relation will appear in the primary display window;

- This frame does not offer editing features or a window showing topics; if the user wishes to access editing features from this frame, the selected relation and its text model must be moved to The Resolver frame where it may be modified. Another option for editing is to use the mouse option [Restate this old relation], reached by clicking-right on an existing coherence to modify; see Section 15, "System-wide Mouse Options."

## 12.2 A Walk-through of the Conflict Clearance Frame

The user can compare the text models of two or more conflicting relations, and to change the mesh status or remove any of the existing or proposed relations. The Conflict Clearance Frame also allows the instating of an illegal relation into the mesh.

Typically, the user clicks on one of the existing relations and chooses [Show a model] to see a text model in the secondary display window. The text model for the proposed relation is in the primary display window, and the user can compare two text models for two relations. The user may also click on any topic and choose [Explore] or [Give a brief description].

The user may choose to eliminate the conflict by converting either a proposed or existing relation into an analogy or model. This is done by clicking-right on the relation and choosing [Convert to an analogy] or [Convert to a model]. Another way to eliminate a conflict is to click right on an existing relation and choose one of the options: [Mark relation as invalid], or [Merge new relation with this relation]. Or, the user may click on the proposal and choose [Abandon this proposal]. Each of these options eliminate the conflict with the current proposals by altering their mesh status.

Any of the relations in either relation window of this frame can be discarded by clicking-right on them, and then clicking on the pop-up window option [Mark relation as invalid]. If the current user was responsible for creating the original relation, s/he will also be queried as to: [Kill it forever], or [Merely mark unaccepted]. The former choice expunges the relation permanently from the mesh; the latter merely marks its status.

On the other hand, the user may click on an existing relation and choose [Merge new relation with this relation] to merge it with the proposed relation.

After each of these options, the user may click on [Recompute conflicts] in the command menu. If a conflict remains, this frame will display it. If the conflict is resolved, the user will be returned to the previous primary frame, usually the Write Watcher.

A conflict may be resolved by clicking on either of the existing conflicting relations and choosing the pop-up window option [Resolve this conflict]. The user is taken to the Resolver frame; see Section 12.

Finally, the user may decide to instate the conflicting proposal. In this event, the command menu option [Accept new relation as is] should be selected. The conflicting coherences will co-exist in the mesh.

## 12.3 Command Menu

The Conflict Clearance frame lists all conflicting relations and offers the user a number of ways of handling the conflict by mouse options. Additionally, the command menu offers three choices:

### 12.3.1 [Recompute conflicts]

This option determines whether the relation is in conflict after changes have been made. For example, if the user decides to use a mouse option to mark an existing relation as invalid, or change it into an analogy or a model, conflicts are recomputed with this option.

### 12.3.2 [Leave]

This option allows the user to abandon this frame and return to the last window. As the user tries to leave this frame, a pop-up window will prompt the user to handle the proposed relation. The choices are: [Instate], [Abandon this proposal], and [Merge with original]. For detail on instating, see Section 9.2.4 and 9.2.5. If the proposal is abandoned or merged, the user returns to the previous frame.

### 12.3.3 [Accept new relation as is]

This option instates the proposed coherence in the mesh even though it is in conflict with existing coherences in the mesh.

# 13 The Resolver Frame

When a conflict arises between a proposed relation in a relation window and existing coherences, the user may click on one of the existing coherences and choose the pop-up window option [Resolve the conflict]. This choice takes the user to The Resolver. The user may only be brought to a new Resolver frame in the event of a conflict. Previous Resolver frames may be returned to from another frame by pressing SELECT-META-R. Previous Conflict Clearance frames may be returned to from The Resolver by pressing SELECT-META-C.

With The Resolver, the user may work on the content as well as status of a proposed relation and the existing conflicting coherence, whereas in the Conflict Clearance frame, only the status of the relations can be modified.

When a proposed relation and an existing coherence conflict and they are moved to The Resolver, both are treated as proposals. The existing coherence is also given a temporary proposal form, if changes are made to it in The Resolver.

If the user has a conflict between one proposed coherence and an existing coherence, and moves to The Resolver, there are now four forms which may conflict:

- the original existing coherence;
- the original proposal;
- the new proposal;
- (if the original, existing coherence is changed) the new form of the existing coherence.

During the resolution process, all are treated as transitory forms. After the user modifies the proposals, they may be instated or abandoned. The proposal form of the existing coherence is untouched unless the user explicitly removes it from the mesh. If the newly instated version of a proposal conflicts with its original form, this conflict must be handled by resolving the conflict or by replacing the original proposal form with its new proposal form. A useful way of monitoring how many proposals exist for either the original proposal or the original existing coherence is the [Describe] option (see the Section on the Command Menu for more detail).

There are two kinds of resolution. Local resolution is a resolution between the two proposals currently in The Resolver. A local resolution occurs when the proposals in The Resolver no longer conflict, but there are still coherences in the mesh with which one or both of them conflict. If there are no conflicts between the two proposals or in the entire mesh, a global resolution is possible. If a local conflict exists, neither local or global resolution can occur and the options are not offered to the user. The user may abandon proposals in The Resolver with the option [Leave], but typically the user stays in The Resolver until all unresolved conflicts are corrected.

The Resolver frame is divided into three sections: left, central, and right. Each side, left and right, serves one of the two proposals in parallel. Each has its own command menu, topic window, and display window appropriate to the proposal for that side. The Resolver has three command menus ("Left Side Commands," "Central Commands," and "Right Side Commands"); three topic windows which display the topics for each proposal ("Topics solely in left," "Shared Topics," and "Topics solely in right"); and two display windows. The left and right command and topic windows serve each side appropriately and will differ in the commmands and topics that they display depending upon the proposal they refer to. The central command and topic windows refer to more generally to both proposals. When topics are shared between two

proposed relations, they appear in the central topic window ("Shared topics") and, similarly, commands which apply to both sides appear in the central command window.

## 13.1 A Walk-through of The Resolver

The Resolver allows the user to manipulate new proposals and existing coherences as if they were all proposals. This means that each proposal has two forms: its state before coming to The Resolver, and its altered state. If the new, altered state is instated, the original state must be handled. When the user chooses to [Modify] a proposal and then chooses to [Update with changes], the user creates a new version of the proposal. After creating this updated version, the user will have to explicitly indicate whether the update or the original proposal will serve as the current proposal.

Whenever a text model is altered in the editing frame, the old version is retained. If there is more than one text model for any one proposal, a pop-up window will list the different text models indicating their differing perspectives or emphases. The pop-up window lists a fragment of the different proposed models and the user can view the full text models for those proposals by choosing one. The user may choose one model of the proposal, view it, and then choose another and view it. For instance, if three text models have collected for one proposal, when the user chooses to update the text model of the proposal (again) and the pop-up window appears, there will be four versions to choose from. When the user decides which of the text models to instate for the proposal, the user clicks on it making it appear in the appropriate display window, and exits the pop-up window by moving the mouse away from the pop-up window. The selected text model appears in the appropriate window and will be assigned to the current proposed relation for that side.

The user may modify proposals (proposals for relations or existing coherences) by accessing an editing window with the [Modify] command; or, by changing the mesh status of the proposal by choosing either the [Analogy] or [Model] option.

There are several ways to alter the topics which comprise the proposals. The user may add or take away topics in the editing window that appears with the [Modify] command. Another option which is unique to this frame is the [Substitute] option. This allows the user to change the topics of a proposal by replacing a current topic with another topic which is analogous to it.

The [Undo] option allows the user to choose a collection of topics that was previously used in a proposal during the current resolution process. This option returns the topic window to a previous state, thereby returning the elements of the proposal to a previous condition.

Once the proposals are modified (their topics and possibly their text models) and they do not conflict, they can be instated. If the original existing coherence is changed and instated, it may conflict with its old form. There are two conflict resolution commands in The Resolver: [Local Resolution] and [Global Resolution]. Unless one of these options appears, the user knows that a conflict still exists between either the proposals currently on the screen, their original forms, or the other coherences in the mesh.

The user may examine where the conflicts lie: [Describe] provides a brief listing of the proposals and relations involved; [Examine] is used to return to the Conflict Clearance frame to display the conflicts as described in Section 11.

Once the user is in The Resolver, the frame will remain until all conflicts are resolved, unless the user chooses to [Leave]. As the user leaves this frame, pop-up windows will show which proposals have not been handled and allow the user to instate, abandon, or merge them; see [Leave], below, for further details. If the user tries to instate a proposal which still conflicts, the conflict resolution process will begin again. The user may also abandon the proposal at any time with the [Abandon] option.

## 13.2 Command Menu: "Left" and "Right Side Commands"

Each "side" of the command menu offers identical options, although each side may not offer all of the options at any one time, depending upon circumstances.

### 13.2.1 [Describe]

At any time in the resolution process, this option will list the models that exist for any relation and whether there is any current conflict with other proposals or relations.

### 13.2.2 [Modify]

To resolve the conflict by changing the topics to either proposed relation (and possibly to modify the text model), click on [Modify] in the left or right command menu as appropriate. A temporary and compact text editing window will appear; see Section 9, Write Watcher, for more detail on how to use an editing window. After the text model and/or the elements of the relation have been changed, the user chooses [Update with changes]. This choice creates a new proposal consisting of a relation and its text model. After a proposal is altered with [Modify], the new topics automatically return to the appropriate topic window in The Resolver. Every time the user alters the text model of the proposal, a new version is stored. If there exists more than one text model for the just-modified proposal, the user must choose between

them; a pop-up is provided to make the choice.

Note that in some versions of THOUGHTSTICKER, the most-recent text is not displayed upon returning from [Modify] and the correct one must be chosen from a pop-up displaying these text models; also, all of the text models may be retained and must be eliminated by clicking-right on a relation and choosing [Handle Models].

### 13.2.3 [Undo]

This option will return the topics of the current proposal to the previous state they were in. If a change has been made where new topics were added or removed, [Undo] will return to the previous state.

### 13.2.4 [Model]

This option converts the proposal into a model. After choosing this command menu option, a pop-up window prompts the user to choose which topic or topics in the proposal shall receive the current text as a model. After the proposal is turned into a model, the original form of it must be handled (eliminated, merged, etc.).

### 13.2.5 [Analogy]

This option allows the user to turn a proposal into an analogy. After the proposal becomes an analogy, its original form must be handled, and the user is returned to The Resolver; see Section 13, Analogy, for information on the process of making analogies.

### 13.2.6 [Deny]

This option allows the user to deny the validity of the current proposal (as always, for the particular side). Since the proposal in The Resolver may have an original form, the following pop-up window choices are offered:

- [Yes, the original should be denied], marking it unaccepted;
- [No, the original stands].

### 13.2.7 [Attend to Self]

After modifying one of the proposed relations in The Resolver and updating it with changes, there may be a local conflict between a proposal and its original form. [Attend to Self] will appear in the appropriate command window, indicating that the updated form of the

proposed coherence conflicts with its old form. The possible actions are:

- [Deny original form]: Mark original unaccepted.

- [Adopt new form]: Replace the original proposal with the new form of the proposal. A pop-up window will inform the user that the new proposal is merging with the original relation. It will prompt the user to replace the original topics with the new topics, or not; this gives the user the option of using the old topics with the new text.

- [Try to resolve the conflict]: Bring the old proposal and the new form of the proposal into The Resolver. If there are global conflicts aside from the current local conflict, these must be handled after the local resolution.

### 13.2.8 [Examine]

This option will take the user to the Conflict Clearance frame where the user can view the conflicting relations and the proposed relation, as well as their text models. This option appears only if the proposal has conflicting coherences in the mesh. See Section 11, Conflict Clearance Frame.

### 13.2.9 [Substitute]

The user chooses this command menu option in order to replace a topic in one of the proposals with an analogical topic. A pop-up window prompts the user to choose the analogy from which the new analogical topic will be taken. All of the analogies in the mesh which have one of the current topics as elements are listed. After the user chooses the desired analogy, another pop-up window lists the possible current topics that may be replaced. A pop-up window then lists the topics from the selected analogy and prompts the user to choose which of them should replace the selected current topic. After choosing the replacement analogical topic, the replaced topic leaves the topic window and the new analogical topic is added.

## 13.3 Central Command Menu: "Central Commands"

The Central Commands menu may offer three options when they are appropriate:

### 13.3.1 [Leave]

This option takes the user back to the previous frame. If proposals have not been handled, a pop-up window prompts the user to handle them.

### 13.3.2 [Local Resolution]

If, after modifying one of the proposed relations, the new proposal does not conflict with any other proposals, a "local" resolution is possible; the system has determined that there are no conflicts between the newly-proposed relation and the other proposed relation. Clicking on [Local Resolution] will resolve the proposals in the current Resolver.

### 13.3.3 [Global Resolution]

This command appears when there are no local conflicts and no conflicts with existing relations in the mesh. If the user chooses this option, a pop-up window will prompt the user to make each proposal into a coherence, analogy, or model. The pop-up window shows the proposal that it refers to.

If the user chooses to make a coherence, an "emphasis" is requested in another pop-up window. The emphasis refers to the topic or topics which are the focus of the explanation in the text model. This is used to indicate to the tutorial strategy the appropriatness of the text model to explain a particular topic.

The user is then returned to the Conflict Clearance frame. From there, the user chooses the command menu option [Recompute Conflicts]. Since there were no conflicting relations (i.e., global resolution was possible), the user is returned to the Write Watcher. The new relation which originally conflicted with an existing relation will appear in its modified form in the topic window along with the topics which are its elements.

### 13.3.4 [Separate Topics]

This provides a quick bifurcation of all or some of the topics in common without going through the process of specifying new names or details of the analogy. A pop-up window queries which topics are to be separated.

### 13.3.5 [Combine Topics]

This provides a method to combine all or some of the overlapping topics into a single topic. This is particularly relevant in the case of mandatory unqualified condensation. An ostension of the topics to be combined is made with an analogy formed as in [Condense] to create a single topic to replace the topics of the ostension.

## 14 Analogy Creation

An analogy in THOUGHTSTICKER consists of the objects rendered analogous (termed the "arms" of the analogy) and the point (or "diamond" of the analogy). The diamond and arms are collectively referred to as the elements of the analogy and determine its structure. The significance of the analogy, as described in Conversation Theory, is captured by the "similarities" and "differences" of the elements. In THOUGHTSTICKER, these are currently represented by "constructs" which allow the user to ascribe modifying attributes to the elements of the analogy. Constructs are described in detail in Section 14.

The diamond, if represented as a mesh object such as a topic, can be considered as a generalization or "higher-order" representation of the arms. In some cases, the diamond may not be represented by a mesh object. For example, THOUGHTSTICKER maintains an analogy between any topics which share the same name. If necessary, THOUGHTSTICKER will create such an analogy simply to note that relation if no analogy linking the topics is present. Such an analogy would have the visual representation § -> cat <=> cat <=> cat † if there were three topics with the name "cat." These trivial analogies may be identified by their [Brief Description] which will specify the similarity as "SAME-NAME."

### 14.1 Beginning an analogy

Analogies may be created in THOUGHTSTICKER in several ways. These differ in the manner in which the analogy is initiated and how the possible elements of the analogy are delineated. In most cases, the user structures the analogy by specifying the diamond and arms; for topic bifurcation and condensation of ostensions, the diamond and arms are determined by the operation itself, as detailed below. After the skeleton of the analogy is completed, the user may use the constructs facility to clarify the meaning of the analogy. The various ways of initiating the construction of an analogy are as follows:

- The primary and most direct method to begin an analogy is through the same authoring mechanism used to create coherences. As noted in the section on the Write Watcher, the author is given the option of instating a relation as a coherence (if it has no conflicts) or as an analogy or model. In this case of analogy, the possible elements of the analogy are those the author has previously indicated for the relation.

- The user may invoke the [Environment] command and then the [Make Analogy] command from the pop-up menu. In this case, the possible elements are chosen from all the mesh objects displayed in the current frame. This permits analogies to be made between coherences or even other analogies.

- By clicking-right on a relation and selecting the [Convert to analogy] command on the pop-up menu, the user may make an analogy out of an existing relation. The possible elements are the elements of the relation. The original relation is unchanged by this action.

- During conflict resolution, a conflicting proposal can be removed from the conflict by instating it as an analogy, instead of attempting to instate it as a coherence. The elements are drawn from the elements of the proposal.

- When a topic is bifurcated, an analogy is established with the original topic as the diamond node and each of the new topics as an arm. This differs from the previous methods in that the user specifies the number of arms and then creates new topics to occupy them. The user is also prompted to specify any new names for the new topics; as a default, the system will transfer the name of the original topic to the daughter topics and give an additional name distinguished with one or more symbols made from single-quotes. Thus, if the topic " child " is bifurcated into two topics, they will have the names " child " and " child ' " on the one hand, and " child " and " child '' " on the other.

- An ostension (grouping of more than one topic) represents a collective topic and, as such, is a transitory form. Condensing the ostension creates an analogy between its constituent elements (as the arms of the analogy) and a new topic as the diamond node, for which the user is prompted. The new diamond node replaces the ostension throughout the mesh.

## 14.2 Structuring an analogy

To structure the analogy, the user must select those topics which are to represent the diamond and arms of the analogy. Only the diamond can be left unspecified; such a structure corresponds to an analogy for which the diamond has not been instated as a topic. The diamond node can be supplied and instated as a topic (if necessary) at any future

time by using the [Provide diamond] command on the analogy pop-up menu. The initial list of possible elements for the analogy are determined by the method of entry into the analogy formation process as described above. The list is displayed to the user in a pop-up menu with choices boxes. The choice boxes provide a variety of functions:

- [Return bold choices]: This completes the choosing process. For analogy formation, the return of more than a single choice for the diamond or one of the arms automatically causes an ostension to be made; the ostension represents that group of elements in the analogy. Only the diamond can be left unspecified, in which case the user must confirm that this is the intention. If an arm is left unspecified, by making no bold choice in the pop-up, the analogy structure is presumed complete.

- [Add topic]: If the list of possible elements from which the analogy is formed does not contain a desired topic, the user may add it to the list by clicking on this choice box. A window is popped-up to enter the name of the topic. As in the corresponding command in the Write Watcher, if a unique topic exists with that name, it is simply retrieved; if no such topic exists, one is created with the entered name; if any ambiguity is present, the user is queried for clarification. The new topic is added to list of items in the menu (the menu may need to be scrolled to make it visible). This choice box does not directly change the structure of the analogy; it merely enlarges the range of possible analogy elements. The timely use of this choice is recommended if the user wishes to create a topic to represent the diamond.

- [Explore Bold]: The topics currently indicated in bold are explored, that is, the tutorial mode is entered for those topics.

- [Brief Description]: Brief descriptions of the bold items are given.

- [Help]: A summary description of the process of analogy structuring is given in a temporary window.

## 14.3 Using the constructs facility

Having completed the structure of the analogy, the significance of the analogy is established using the constructs facility. The user may apply any existing construct to the analogy; create a new construct and then apply it to the analogy; or provide general remarks about the analogy. A pop-up menu is provided with the commands as described below.

At present, THOUGHTSTICKER absorbs the constructs but does not change its behavior depending on their values; this will be changed in future releases. For the moment, they can be considered as an aid to the authoring process in the refinement of the mesh.

### 14.3.1 [Quit]

The analogy is now complete. No further constructs or remarks are to be made and the pop-up is removed.

### 14.3.2 [Help]

A temporary window provides assistance to the analogy process.

### 14.3.3 [Describe constructs]

Each of the constructs previously used by the current contexture is briefly described. The name, documentation, choice type, default value, and choices are given.

### 14.3.4 [Use construct]

A menu of the defined constructs is provided. The name of the construct appears in the menu; the documentation provided for each construct is given in the mouse-documentation line when the cursor box appears over each construct choice. Having selected a construct, a window listing all the elements of the analogy allows each element to be associated with a value of the construct. For example, if the construct is "age" with choice type NUMBER, the user may click next to each element and type in the appropriate number. For other choice types, such as CHOOSE, BOOLEAN, and TRUTH-VALUES, simply clicking on the corresponding value suffices. Finally, clicking on "Exit" completes the assignments; "Abort" undoes the application of the construct. See Section 14 on Constructs for complete details.

### 14.3.5 [Make new construct]

The user is led through the making of a construct to apply to the analogy. The name of the construct, documentation, choice type, and any fixed or default values must be supplied. See Section 14 on Constructs for complete details.Section 14 on constructs.

### 14.3.6 [Make remark]

A temporary type-in window is provided. Any text may be entered to explicate the meaning of the analogy in any way desired. This may be used to add additional information in circumstances for which the constructs facility is otherwise insufficiently general.

### 14.3.7 [Remove construct]

Remove the association of a construct with the analogy. The construct itself may not be removed, as it may be used in other analogies or in other meshes.

## 15 Contextures and Constructs

This section describes in detail the process of creating new contextures and constructs. These concepts form the basis of THOUGHTSTICKER's representation of an individual user and his personal vocabulary or repertoire of ideas, in so far as they are separate from the mesh itself.

Contextures identify the user on a particular occasion. Associated with the author's contexture are personalized names for topics; a history of interaction which is automatically maintained by THOUGHTSTICKER relative to contextures; and constructs. The constructs are detailed information regarding the analogies held in the mesh.

### 15.1 Contextures

A contexture represents a particular user on a particular occasion of interaction with THOUGHTSTICKER. A contexture may relate to one or more meshes and THOUGHTSTICKER maintains its history of the user's interaction relative to a contexture. Strategies for choosing models and new topics are relative to the information stored in the contexture. Also, the display of topic names is performed relative to the current contexture. For these reasons, the contexture is responsible for personalizing the interface for the user.

In THOUGHTSTICKER, a contexture is identified and referenced by a list of names or short phrases, collectively referred to as the "author." The term "author" is used simply to indicate that the contexture is a personal, active agent in the mesh; "actor" or "agent" could equally well be used and contextures are maintained for users who do not add additional material into the mesh.

A list of names is used to facilitate the identification of groups of related contextures. For example, contextures associated with tests of holist and serialist learning strategies could be identified by a list such as ("AMTE training test", "Holist") and ("AMTE training test", "Serialist"). Alternately, ("AMTE", "Training", "Test", "Holist") and ("AMTE", "Training", "Test", "Serialist"). There is no limit in principle to the number of such texts that can be used to distinguish

contextures. In the current release, up to five groupings can be made. When a contexture is first created, the user specifies the "author" by providing one or more such names.

Each contexture is associated with a collection of contextures for use in analogy creation. THOUGHTSTICKER maintains an association between all related contextures so that they may be stored separately but recalled as a group when needed.

Strategies for THOUGHTSTICKER to make choices on behalf of the user are formulated as lists of comparison criteria, or "predicates." Given a variety of alternative choices or actions (for example, to choose a text model to explain some topic), THOUGHTSTICKER endeavors to place the list of alternatives in order of comparative worth, relevance or importance.

THOUGHTSTICKER applies each predicate in a specified order until a preference among alternatives is obtained. By varying the order of predicates, radically different training and authoring strategies can be used within THOUGHTSTICKER. The system governs the strategy from a specification of predicates that are to be used and determines the relative importance of each of the criteria by specifying the order in which they are applied. The user specifies the predicates to be used and their order for each decision strategy by means of "Choose variable values" pop-up windows (Section 4.3.5).

The predicates maintained are generally self-explanatory and only those needing explanation are mentioned below. They are set-up by the user when the contexture is defined. Predicates are maintained for the following situations:

1. Ordering relations: In the tutoring/training mode of the Aim Auditor, THOUGHTSTICKER tries to determine the best relation to use in explaining a group of aim topics for the command [Why is]. In a few of the predicates, "overlap" refers to the overlap between the topics currently aimed for and the topics associated with the possible relation to be used for tutoring.

2. Choosing topics: In the training mode the Aim Auditor, the user has the option of asking THOUGHTSTICKER to provide a new topic to aim for with the command [What Next]. The predicate "want better relation" uses the predicates for ordering relations just described in order to determine a topic which has a relation of best merit by those criteria. "More general" or "less general" refers to topics linked by analogy, where "more general" moves from an arm to a diamond and "less general" is vice versa.

3. Topic disambiguation: If two topics share a common name, THOUGHTSTICKER may use predicates to provide a default choice between them. "More general" or "less general" refers to topics linked by analogy, where "more general" moves from an arm to a diamond and "less general" is vice versa.

4. Saturation ordering: In the saturation operation, a list of topics is used as a basis for suggesting new relations that could be instated legally in the mesh. The exhaustive search algorithm used to generate the possible suggestions may be guided by ordering the list of topics to be saturated according to these predicates. The same criteria are used to select topics from the whole mesh for saturation with the 'Suggest Topics' command option.

5. Model ordering: If more than one model could illustrate a particular topic or relation, these predicates provide a choice. The most common of the available predicates, dont-want-recently-executed, exhibits the models in sequence.

The strategies and the number and variety of predicates within each strategy may change in future THOUGHTSTICKER releases. Presently, THOUGHTSTICKER supplies 32 predicates.

The predicates used and their order may be changed at any time by choosing the 'Edit contexture' command in the 'Environment' pop-up menu. For each of the maintained strategies (ordering relations, choosing topics, etc.) a pop-up window will list the possible predicates that could be applied. Predicates may be removed or added to the list used by the contexture. Brief descriptions of the action of each predicate is provided in the mouse-documentation window. Having chosen the desired predicates, a second window is provided with the list of chosen predicates with choice boxes numbered sequentially.

To choose, for example, "want better relation" as the most important criterion, the user clicks on the box under column 1 next to "want better relation." If a predicate is not assigned a priority, THOUGHTSTICKER gives it a default priority; note that this does not permit predicates to be dropped completely.

A help facility is provided on the first pop-up window to review the detailed procedure.

## 15.2 Constructs

Constructs are used to permit the user to supply similarity and difference information in the construction of analogies. In the present release of THOUGHTSTICKER, this is primarily accomplished by attaching modifying descriptions to each of the elements of the analogy. For example, two individuals may differ in age, one being young and the other old. We might refer to this construct as the "age" construct, with possible values of "young" and "old" (as noted below, a neutral value or set of values is often useful, such as "ageless", or "does not apply"). A different way of specifying the age might be to give the age in years. In this case the values of the "age" construct could be any number. Note that the construct captures both the similarity and difference necessary for the proper specification of an

analogy. In this case, the similarity is "age-ness" and the differences are specified by the values.

By allowing the user to re-use the identical construct (rather than one which is superficially similar), THOUGHTSTICKER will permit the development of flexible analogical reasoning tools. Other examples of constructs in the THOUGHTSTICKER sense would be:

- "location" with values of "no where", "no where special", "ubiquitous", "Atlantic coast", "Pacific Coast", "Great Britain" and "Continental Europe";

- "part of speech" with values "word", "noun", "verb", etc.;

- "truth value" with values "true", "false", "maybe", and "undecidable";

- "name" with the value being the name of the element.

Constructs may be created as needed during the construction of an analogy or by using the [Make an Analogy] option on the "Environment" command present on most primary THOUGHTSTICKER frames. Each construct consists of:

1. A descriptive name: This can be of any length, but a short phrase is best since the phrase will be used to identify the construct in other windows. The descriptive name should define or encapsulate the nature of the construct.

2. Documentation: This can be a longer phrase, but its length should not exceed the width of the screen. This will appear in the mouse documentation line whenever the construct is being used.

3. The type of choice: This is one of the input mechanisms supported by "Choose variable values" of the Symbolics. THOUGHTSTICKER does not use all of the possible types; those used are described below.

4. A default value: If desired, the user may supply a default value. If the user does not specify a default and NIL[1] is a possible value of the construct, then the default is set to NIL.

For the "CHOOSE" choice type only, a list of the possible choices must be provided (see below).

As noted, the construct represents both similarity and difference terms in the analogy. The user must select the type and values of the construct accordingly. In Conversation Theory, the diamond node of an

---

1. As used here, NIL is the symbol for "nothing."

analogy can be seen as representing in topic form the similarity of the topics made analogous (the arms of the analogy). The construct is a mechanism for naming or modelling that similarity. Therefore, each construct must have a range of values that can be applied to all the elements of the analogy, not just the arms. For example, if the values applied to the arms represent the differences between them, the value assigned to the diamond must capture the similarity between the values or indicate the inapplicability of the value. Examples will be given below.

The precise method of applying a construct to an analogy is detailed in Section 14 on Analogy. The choice mechanisms used in the THOUGHTSTICKER constructs facility are referred to by their Symbolics System names (as detailed in the Symbolics documentation on "Choose variable values" windows) and are described in the following subsections. Each paragraph contains the name of the Symbolics mechanism for making a particular type of choice. The user chooses which choice mechanism is appropriate to the particular construct, thus allowing for maximum flexibility in their definition. For example, if the user wishes to define a construct from a set of choices such as "big", "little", etc., then the CHOICE mechanism is used; if the construct should consist of an individual's age, then the NUMBER mechanism is used; and so forth, as explained below.

### 15.2.1 CHOOSE and CHOOSE-MULTIPLE

These permit a choice among a fixed set of possibilities. When the construct is created, the user must supply all the possible values for the construct. When the construct is used, the various possible values will be listed and the user makes the choice with the mouse. For example:

> "The big plant and the small plant are both roses; the big one is a yellow rose and the little one is a red rose."

This text could be a model of the analogy
{roses -> big plant <=> little plant }. The topic "roses" is indeed the similarity between the two plants. There are two constructs that might be supplied:

- Size: The short name could be "size"; the documentation might read "Things may differ in size." The values of the construct needed for the arms of the analogy are "big" and "little"; the value for the topic "roses" depends on the user's perspective. Two extreme possibilities are

  * "Does not apply": Although "roses" captures the similarity between the two plants, there is no value that captures the corresponding similarity between "big" and "little". The topic "roses" cannot have the attribute of "size" directly associated with it.

* *"Has a size" or simply "size": The user recognizes that "roses" have a quantifiable attribute called size. This identifies the name of the construct with the value to be assigned to the diamond element.*

- Color: The values to be applied to the arms might be "yellow", "red", "white", and "black". Note that the construct need not be restricted to the values needed at the moment (only yellow and red). Again the value that should be assigned to "roses" may vary from "have many possible colors" to "does not apply."

CHOOSE provides an exclusive choice; only one of the specified sub-choices can be chosen. CHOOSE-MULTIPLE allows more than one of the sub-choices to be associated with the element.

### 15.2.2 NUMBER and NUMBER-OR-NIL

These choice mechanisms can be used when the values of the construct are expected to be essentially arbitrary numbers. The choice mechanism will reject any input that is not numerical. The choice type NUMBER-OR-NIL is useful when, for example, the diamond node cannot have a definite number assigned to it. The initial value of the number will be the default choice specified during the creation of the construct.[2] For example:

"John and James are brothers, aged 34 and 53." The analogy { brothers -> John <=> James } could use the construct "age" of type NUMBER-OR-NIL, with values of NIL, 34, and 53 respectively.

"Jill and Jane are sisters, aged 23 and 17." The analogy { sisters -> Jill <=> Jane } could use the same construct, with values of NIL, 23, and 17.

As noted above, the use of the same construct in both analogies make possible certain forms of analogical reasoning, such as "Jane is to _____ as John is to _____," or, "Find analogies in which Jane is related to someone via a construct which is the same as one in which John is related to yet another person."

### 15.2.3 STRING, STRING-OR-NIL, and STRING-LIST

This is a liberal form of choice which permits strings (text) of an unrestricted nature to be entered as the value of the construct. STRING-LIST is distinguished from STRING by allowing a list of separate

---

2. If this is NIL, then NUMBER will behave similarly to NUMBER-OR-NIL unless the value of NIL is changed; it cannot be changed back. Use NUMBER-OR-NIL for the general case.

strings to be entered (separated by commas). For example:

"My children are named Heather Anne and James Craig William." The analogy § children -> Heather Anne <=> James Craig William † could have the construct "name" and choice type STRING with values "" (the so-called "null" or "empty" string, equivalent to "does-not-apply"); "Heather Anne"; and "James Craig William." Alternately, the choice type could be STRING-LIST with values of "", ("Heather", "Anne") and ("James", "Craig", "William") respectively. In this latter case, one choice could be "Heather", "Anne" or "Heather Anne"; in the former, the choice relating to the same child could only be "Heather Anne".

This particular analogy could have been handled with a construct of the CHOOSE choice type but it then could not have been used to specify the names of any other individuals. With the CHOOSE choice type, each such naming would be separate; with the STRING-related mechanisms, there is automatic linking of new values associated by the name of the construct.

An alternative way of handling the case of the empty string is to use the STRING-OR-NIL choice type, which permits the entry of NIL as a value.

### 15.2.4 BOOLEAN and TRUTH-VALUES

These choice types are provided to indicate (contexture specific) labels for truth and falsehood. The Symbolics standard choice type BOOLEAN permits simple two-valued YES and NO indicators to be used as a construct. The lack of a sensible neutral value makes it difficult for this choice type to capture the essence of the similarities and differences of any analogy. It can be used to provide information of a peripheral nature. For example:

"Fred and Mary are very good friends." If the analogy were § friends -> Fred <=> Mary † the construct "Female" could possibly be used with BOOLEAN values of NO, NO, and YES respectively.

Future THOUGHTSTICKER releases will support more extensive multi-valued logic systems. In the present release, the choice type TRUTH-VALUES (not a standard Symbolics option) offers a wider range of values than BOOLEAN, with a range of possible neutral values. The possible values are TRUE, FALSE, MAYBE, UNDECIDABLE, INAPPROPRIATE, and HAS-ALTERNATIVES. As examples:

- "This sentence is false" cannot have the truth values of true, false, or maybe. We may describe it as an UNDECIDABLE proposition or one for which truth values are INAPPROPRIATE.

- "Lisp is a famous computer language with dialects including Maclisp and Interlisp" could be a model of §Lisp -> Maclisp <=> Interlisp†. If a construct called "Used

at M.I.T" were used with the choice type TRUTH-VALUES, the values chosen might be HAS-ALTERNATIVES, TRUE and FALSE, respectively. The topic "Lisp" cannot appropriately be used in this analogy with any truth value corresponding to "Used at M.I.T." but its alternative forms, as given by the analogy itself, represent particular dialects and can have truth values within this context.

## 16 System-wide Mouse Options

For purposes of reference, the following section provides a breakdown of the menu selections of THOUGHTSTICKER which are accessed through mouse choices and appear in temporary pop-up windows. These menu selections are not continuously displayed in the Command Menus, but are retrieved upon the user's initiation. This breakdown provides a description of the menu selections as well as an indication of how to access them. In some cases, the particular selections that are offered depend upon the user's history; for example, if the user clicks left on a relation which already has a text model, the selection [Create a text model for this object] will not appear. The sections below will describe the user options that are likely to appear under various circumstances. In some instances, after the pop-up window appears, clicking left or middle will offer different variations of the option.

### 16.1 Clicking on Analogies

If an analogy does not have similarities and differences as specified by the constructs facility and if the analogy was created by the current contexture, then the option [Establish Point] appears on the pop-up window. This activates the constructs facility.

### 16.2 Clicking on Models

By using the [List Anything] option in the [Operations] pop-up menu from the THOUGHTSTICKER Central Window (SELECT-T), one may list the models in the mesh. By clicking right on the model one may:

- [Execute Model]: The model is shown or executed.
- [Reuse this Model]: The model may be attached to a new topic or relation.

## 16.3 Mousing Topics versus Relations

Note that there is a slight trick to having the mouse distinguish between relations and the elements which make up the relations. When pointing directly at a topic, a box will appear around the topic only. When pointing between the topics, the box will appear around the relation containing the object; for example, the entire coherence. The box indicates what would be chosen if a button is clicked.

## 16.4 Clicking on Topics or Relations

When the user clicks any mouse button on topics or relations, the following options appear in a pop-up window:

### 16.4.1 [Give a brief description]

Shows the user a tally of the models and relations that the selected object appears in.

### 16.4.2 [Remove from window]

Removes elements from their respective windows. If the user clicks on a topic or a relation in any window and then chooses this option, the topic or relation will be removed from the topic window or the relation window, respectively.

### 16.4.3 [Move neighbors to list]

If the user clicks on a topic and chooses this option with click-left, the topics which are in relations containing the selected topic will be added to the topic list; if the user clicks on a relation and chooses this option with click-left, all relations which share topics with the selected relation will be added to the relation list.

Clicking-middle first clears the respective list and then places the appropriate elements into the window.

### 16.4.4 [Create a text model]

This option appears only if no model presently exists. Gives the user a temporary window called "Providing model statement editing commands," a window related to the Write Watcher. In this window the user can create a text model for a topic or relation. See Section 9, Write Watcher.

### 16.4.5 [Show a model of this object]

This option appears only if a model exists. If the user clicks on a topic or relation, a model is displayed.

## 16.5 Clicking on Relations

When the user clicks any mouse button on relations, the following options will appear in a pop-up window:

### 16.5.1 [Move elements to list]

Elements of the chosen relation are moved into the topic list. These elements may be topics, ostensions, or analogies. Clicking-left on this option adds the element, while clicking-middle clears the existing topics and then moves the element to the topic window.

## 16.6 Clicking on Topics Only

If the user clicks any button on topics, the following options appear in a pop-up window:

### 16.6.1 [Move this to list]

Clicking-left on this option moves the selected topic to the topic window. Clicking-middle replaces the topics in the topic window with the selected topic.

## 16.7 Clicking Left on Topics Only

Clicking left on topics will provide these options:

### 16.7.1 [Move all relations to relation list]

Moves all relations that include the selected topic, accepted or not accepted, to the relation window. Clicking-left on this option adds the relations to the relation window. Clicking-middle replaces the current relation list with the new relations.

### 16.7.2 [Build a word phrase]

Uses the selected topic as the start of a topic word phrase (for example, "conversation theoretic ideas"). The user clicks left on the successive words of the phrase, and clicks middle or presses END to end the phrase. RESUME will cancel the transaction at any point. The word phrase is visible in the lower left corner of the display window.

### 16.7.3 [Explore]

Moves the user to the Explorer, to be used for the learning of the topics in the most current topic list. See Section 8.4, The Explorer.

### 16.7.4 [Replace topic by analogous topic]

Allows the user to replace a topic (in the topic window of the Aim Auditor or the Write Watcher) with another topic that is analogous to it. The user clicks left on a topic, chooses this option and follows a series of pop-up windows in order to replace the selected topic with a related topic from an analogy. Both the selected topic and the replaced topic must be topics in the same analogy. A pop-up window displays the analogies which have the selected topic in them. The user picks the analogy from which to choose the analogous topic; and then picks the topic in that analogy that is to be placed into the topic list. The original topic is replaced.

## 16.8 Clicking Right on Topics and Relations

### 16.8.1 Prune this

A prune of the object (topic, ostension, or relation> is displayed. The user can control the depth of the prune. The prune provides both relations and a list of the topics in the relation so that the user may expand the prune through one of the topics, thereby viewing a larger prune. The user may click on any topic, relation, or portion of the prune. This expands the prune through the portion clicked on. The right, middle and left mouse buttons provide different viewing options for the construction of prunes with all relations, coherences only, and for exploring topics and relations. Once a prune is constructed, the user may return to the previous prune screen or return directly to the previous THOUGHTSTICKER primary frame. (The mouse documentation window provides explanations of these options.)

### 16.8.2 [Handle model]

Allows several ways of manipulating a model. By clicking on this option, the user receives another pop-up window with the following choices: [Show a model], and [Create a text model]. These options also appear in the general case in a pop-up window when the user clicks (left or right) on topics or relations. The new options offered under [Handle model] which appears by click-right are:

- [Approve models]: Allows the user to approve (retain) or disapprove (remove) already created models in the mesh. A pop-up window shows a fragment of existing models for the topic or relation which the user clicked right on, and the user chooses those which are acceptable. In the [Approve models] pop-up window, the user has several options which can be explored by means of clicking on an exit box:

  * [Show the Model]: Show the model before accepting it.

  * [Specify Emphasis]: Examine or change the emphasis of a certain model.

  * [Reuse Model]: Use the substance of the model as a model of another topic or relation. This may be used for either text or non-text models.

  * [Edit a model Text]: If the current contexture created a model, it may in some circumstances be permitted to edit the text. The text may not be edited if other contextures have seen the model in its earlier form. This may be deleted by setting the variable *bypass-other-contexts* to T. In addition, if the variable *time-limit* is a number, (rather than the current default nil), the model cannot be edited if *time-limit* seconds have passed since its creation.

- [Inspect]: This option gives the knowledgeable user the ability to look in finer detail at the structures behind THOUGHTSTICKER objects and how they are implemented in Symbolics flavor system. Choosing this option takes the user to the Symbolics INSPECTOR frame.

## 16.9 Clicking Right on Topics Only

When the user clicks right on topics, several options are available in a pop-up window:

16.9.1 [Move accepted relations to relation list]

Moves all accepted relations which include the topic that was originally clicked on to the relation window. It is distinct from the mouse option [Move all relations to the relations list], which moves accepted and unaccepted relations to the list. Choosing this option with the left button moves all relations that include the selected topic to the relation list. Choosing this option with the middle button replaces the current relation list with the relations that include the selected topic.

[Mark topic as understood]: Marks the topic as understood. (Marking the topic as understood will effect which relations the system suggests that the user aim for next with the [What next] command and which relations the user views first with the [Why is] command, in both the Aim Auditor frame and the Explorer frames.)

16.9.2 [Mark topic as not understood]

Marks the selected topic as not understood.

16.9.3 [Handle name]

Allows several ways of manipulating names:

- [Remove an old name from this topic]: Cancels a previously created name for the chosen topic.

- [Add a new name to this topic]: Creates a new name for an existing topic. (This option permits each contexture to use a personal vocabulary.)

- [Choose primary name]: Change the usual name of the topic to appear on the screen. Since a topic may be referred to by more than one name, this allows the user to specify which previously used name will be used in the future in the topic and relation lists.

[Bifurcate]: Separates the selected topic into two or more analogical topics. The user creates an analogy between two or more similar topics and provides constructs for each arm of the analogy; see Section 13, "Analogy Creation" and Section 14 on Constructs.

## 16.10 Clicking Right on Relations Only

When clicking right on relations only, the following option is available:

### 16.10.1 [Convert to an analogy]

Converts a relation model into an analogy. A series of pop-up windows allows the user to define the elements of the analogy and its constructs. The old relation is left intact and must be manually removed from the mesh if desired. See Section 13, "Analogy Creation" and Section 14, on Constructs.

### 16.10.2 [Convert to a model]

Changes a model for a relation (whether analogy or coherence or ostension) into a model for a topic. From a pop-up window, the user chooses one or more of the topics from the relation. The models of the relation become models for the indicated topics. The original relation is not affected.

## 16.11 Clicking right on Ostensions or Relations

### 16.11.1 [Condense]

Prompts for a topic to represent the ostension at a higher order. Replaces the ostension with the topic where possible.

Any relation or ostension may be condensed to a single topic. For example, the coherence [Cats ⊕ Eat ⊕ Mice] may be condensed to 'house-hold ecology'. An analogy is made of the form {house-hold ecology --> [Cats ⊕ Eat ⊕ Mice]}. The usual constructs may be applied to this analogy. If the object condensed was used in some other relation (the most likely case being an ostension), the user may substitute the new topic for the old object using analogical substitution.

# Chapter 17

# Appendix

## 17.1 THOUGHTSTICKER History

One of the authors (Pangaro) offers the following personal history of THOUGHTSTICKER technology, starting from its first named embodiment in digital machines.

### 17.1.1 The "first" THOUGHTSTICKER Software

A software system running on digital computers and named THOUGHTSTICKER first existed around 1977 at the laboratory of System Research Ltd in Richmond-upon-Thames, Surrey, under the direction of Dr. Gordon Pask. The hardware configuration consisted in a variety of peripheral devices:

1. Text retrieval, in the form of "pidgeon hole" slots with printed paper (computer storage was at a premium and both RAM and disk were entirely taken up with program);

2. Hi-resolution line graphics displays, for the display of the evolving knowledge representation, especially the verification of valid constructions;[1]

3. Manual board for user construction of complete knowledge "maps," with computer-controlled status indicators;

4. A digital computer, the heart of the system, running the software of the THOUGHTSTICKER system.

---

1. These were ARDS display tubes, considered the first mass-produced graphics displays. Some few hundred were produced, and these came to Pask via Negroponte at MIT. They were provided as part of a barter exchange. The display tubes with their driving electronices are now of historical value in the history of computer graphics, and Pask is the source for the artifacts to be installed in The Computer Museum of Boston.

PAGE

MISSING

IN

ORIGINAL

The software for this original system was written in a LISP variant.[2] This THOUGHTSTICKER was limited in complexity of data structure and in performance.[3] One of its major contributions was its originality: it was one of the first and perhaps the first truly domain-free software program. It could represent the knowledge and beliefs of any individual, in any domain, using the identical software. It was also the first program that attempted to display, in a useful and psychologically meaningful, way the topological structure of the knowledge. The system existed with both elicitation and representation modules in 1978.

### 17.1.2 The first micro-based implementation: MTHSTR

The limitations of the original THOUGHTSTICKER were clear to anyone viewing it, and the primary one was perhaps its imprisonment in a completely non-standard, baroque, and incompatible environment of the research laboratory of System Research (literally, and often referred to by those who visited, as the research basement). The obvious path for future exposure and extension was to re-implement the system in some transportable and reasonably-available micro-processor environment. For the moment, the limitations of memory size implicit in micro computers would be set aside for the other benefits, principally their economy and therefore their availability.

Robin McKinnon-Wood, long-time collaborator of Pask and contributor to the Cambridge (University) Language Research Unit, conceived of and coded a database scheme for the BASIC language.[4] This scheme was clever in that it used the string capabilities to emulate what would be

---

2. Both THOUGHTSTICKER and the LISP-like language it was coded in, called LISPN, was written by Robert Newton of the staff of System Research Ltd. LISPN was an interpreter written in assembly code for the Computer Automation LSI machine, one of the first mini-computers available in England. LISPN contained many of the primitives of the LISP 1.5 of McCarthy, but specifically not the lambda operator for function definitions. This would seem a fundamental contradiction to LISP itself, and it was; Pask insisted on it.

3. All system code was swapped from 8" floppy disk, rather inefficiently. The system as whole with its peripherals was a tour de force, considering the era of computing in England and the circumstances of funding.

4. The hardware was produced by Research Machines Limited (RML), and ran a Z80 processor in a custom operating system. This was in 1978-79 and CPM was not yet widely available, at least on machines distributed in the UK. The BASIC was a fairly good implementation of what would become the MicroSoft version of the language.

simple list-processing primitives in LISP. In some 7 pages of BASIC code, MTHSTR, as it was dubbed, contained the basic operations of Lp, the knowledge calculus of Conversation Theory.[5] The means for representing the Lp structure of a coherence in a static database was designed in collaboration with Paul Pangaro, who was a student of Pask at the time and knowledgeable in computer science.[6]

MTHSTR proved that the algorithms of Lp were tractable for small machines and simple programming languages. The implementation was self contained and did not require baroque and often fragile hardware in attendance; a simple micro was enough. However, the difficulties of MTHSTR were clear: small data space, slow execution, and limited functions.

### 17.1.3 PASCAL TSTIK

The Applied Psychology Unit (APU) of AMTE Teddington became interested in Pask's work and funded an effort around 1980 to place THOUGHTSTICKER into the HP1000 systems which they had on-site at APU.[7] The entire project was fraught with difficulties, including but not limited to the problems associated with using contractors not familiar with Conversation Theory, obsolete operating systems, the management of computing facilities, and collapsing research organisations. Despite

---

5. Specifically, MTHSTR included instatement of coherences, Genoa contradiction checking (for certain cases only), Prune and Selective Prune (for most cases), and other utilities such as listing topics, coherences and pruning structures. McKinnon-Wood, an algorithm man from way back, understood the meaning of Turing computability and universal machines. He was not daunted by lack of flexible data types, linked lists and pointers, list operations, and modularity. His cleverness was in performing complex, multi-dimensional parsing using a lexical scheme and one-dimensional string arrays.

6. Pangaro later experimented with moving MTHSTR to the DEC 11 and the IBM 5120, and a translation of MTHSTR into these environments was achieved. However, these efforts were hampered by the impoverished BASIC of those systems. Some further experiments were done in APL and a condense/expand was written in MacLISP but none of these efforts produced a complete THOUGHTSTICKER.

7. Pangaro provided written specifications for the database routines and Lp operations, including a scheme for condense/expand, as extracted from long arguments with Pask himself as to their nature and significance. These specifications were given to a programmer, working for an established Ministry of Defense contractor. The system was later taken over by Peter Clark, a computer scientist and student of Pask, who endeavored to finish it.

this, TSTIK, as it was called, contained a menu of some twenty commands and some features only recently duplicated in significantly different hardware.[8] It was used for a brief time in an exploratory way but was lost when the computing environment in which it ran was superceded.

### 17.1.4 Apple CASTE, a version of THOUGHTSTICKER

MTHSTR itself as a historical artifact first demonstrated THOUGHTSTICKER functions in a micro environment. Significantly, its database scheme became the basis for a system with many cosmetic and functional extensions, running in the Apple ][ microcomputer.[9]

These extensions were made by Pangaro initially in 1982-3, and later by Scott Henderson of the staff of PANGARO Incorporated, under contract to APU and the US Army Research Institute, Virginia (ARI). The system was called CASTE, after Pask's Course Assembly System and Tutorial Environment, a chronological predecessor to THOUGHTSTICKER. The first CASTE was concerned with the structure of materials for learning, and this inquiry led to the knowledge structuring of THOUGHTSTICKER. Therefore, the Apple ][ version of THOUGHTSTICKER was called CASTE to emphasize its authoring and tutorial capabilities. Perforce it contains THOUGHTSTICKER elements.[10] The primary contributions of this version were its ease of use; the completeness of the environment, which contained a modelling facility for a simulation game and a full set of utilities including a text editor, window system, and segmented code; and the fact that it proved the practicality of a complete environment based on Conversation Theory within micro-computers.

One innovation of the system, conceived as a result of collaboration between Pangaro and Dik Gregory of APU, was the use of sentences of

---

8. Included were instatement of coherences, full Genoa contradiction check, pruning and selective pruning, saturation, (crude) condensation/expansion, bifurcation of topics, merging of universes, and a variety of useful utilities.

9. For the record, it should be noted that the Apple configuration required to run this version of THOUGHTSTICKER is not a pure Apple system: it requires a Z80 processor card, 80-column text display card, the CP/M operating system, and, to run efficiently on reasonably-sized but still modest demonstration environments, 256K RAM-Disk cards.

10. Especially instatement of coherences, full Genoa contradiction checking, pruning and selective pruning, and saturation, along with many utilities.

English language text to represent the relation between concepts in the system.[11] These sentences would be provided by the author and are the primary information seen by the student while being tutored.

A "Rules Tutor" for the naval simulation game called HUNKS was constructed. HUNKS involves opposing commanders of fleets of vessels and the ability to command the simulation to move the vessels, fire misiles, and call for information on graphics displays. Gregory used the system to construct both text and simple graphics frames to provide tutorials for learners of rules of the game. The configuration required a second Apple ][, running the HUNKS code and driven by the CASTE software in the other machine. This configuration was completed in 1983.

A variety of test domains have been constructed for this implementation, including a word processing tutor (which contains basic concepts of word processing as well as command conventions and user "HELP" features") and an operational database for corporate documentation, conceived and supervised by Heather Harney of PANGARO Incorporated. The system continues to be extended under contract to the ARI. As of this writing, Pask and his associates at the Centre for System Research and Knowledge Engineering, Concordia University, Montreal, Canada, are adding additional controllers for slide projections and multiple-screen tutorial modes.

### 17.1.5 THOUGHTSTICKER 3600

The purpose of developing the Apple CASTE version of THOUGHTSTICKER was to bridge the time between the demise of the PASCAL version and the planned version to be written in LISP on a hardware configuration of sufficient size and performance to thoroughly test the capabilities of THOUGHTSTICKER in a serious research implementation. APU placed a contract in early 1982 with PANGARO Incorporated to obtain the necessary hardware and expertise in Washington DC, while themselves beginning the procurement process for their own, identical hardware. During the same period, ARI became interested in a large-scale version as part of Pask's contract at Concordia. ARI will share the development by APU by joint collaboration through a NATO sub-panel.

The hardware chosen by Colin Sheppard of APU was the Symbolics 3600, a special-purpose mini-computer which is a hardware engine for running ZetaLISP, a superset of MacLISP, itself derived from McCarthy's LISP 1.5. This system is unsurpassed in performance, features and software support environment. The system came into use in Washington in May of 1983.

---

11. The PASCAL TSTIK also contained this feature at one point, but it was not fully explored there.

The system as it presently exists is as a research tool; that is, it is not yet intended for naive users. The first step of the research program was to implement the basic functions of Lp, with an interface that is "naked" in the sense that all Lp operations are performed manually and all implications of user actions are displayed explicitly. As of this writing, the "naive" version is under development. For APU, THOUGHTSTICKER will be evaluated in the context of the "Intelligent Support System" concept as developed by Sheppard and others of APU. ARI is interested in THOUGHTSTICKER for its possibilities as a system for eliciting and comparing plans of commanders in complex decision making situations.

## 17.2 Undoing and Unhanging

This section describes various means for retracing steps in a sequence and for recovering from mysterious behavior.

### 17.2.1 Cancelling choices

In some cases, a menu choice will start a sequence of further choices that the user regrets. The "Abort" choice box on some menus may not carry the user back to the point desired. The ABORT key on the keyboard will usually cancel the current transaction.

### 17.2.2 Cancelling and waking up

The following are useful for cancelling a previous choice and may also be applied whenever the system is apparently unresponsive or the display appears to be in some way peculiar:

1. Click with the mouse. If the user expects to be able to type into a window and the keyboard appears to be dead, click the mouse over the window. This action is recommended when there is a cursor which is not blinking on the screen. Blinking cursors indicate that the window is listening for type-in.

2. Move the mouse. Some windows refuse to disappear unless the mouse is moved.

3. Use the ABORT Key. This is the mildest and least dangerous form of aborting.

4. Use META-ABORT. This has the effect of waking up the frame.

5. SELECT-ing away and SELECT-ing back. In some cases, selecting a different window and then returning to the previous window can

cause the window to properly redisplay itself.

6. CONTROL-ABORT and META-CONTROL-ABORT. These are more drastic, but properly wake up the system in some cases.

7. Reset the process. This may be necessary in some frames which may interpret ABORT and CONTROL-ABORT in a non-standard manner. The user clicks-right-twice on the mouse to summon the SYSTEM MENU. In the middle column of that menu, under the heading "This window", is the choice RESET. Clicking on RESET causes a confirming window to pop up, saying:
"Confirm: Reset process in window <some name>." Clicking on this tiny window will wake up the frame.

8. If none of these measures avails, singly or in combination, simply SELECT a new version of the window of the same mesh and contexture with SELECT-HYPER-CONTROL and proceed (for example, SELECT-HYPER-CONTROL-A).

### 17.2.3 Output Hold or Window Lock

Under extreme circumstances, it may be possible for a user to issue a complex sequence of commands which "locks" or otherwise confuses the Symbolics window system software. There are a large number of system commands which deal with the window system directly and may allow the THOUGHTSTICKER system to continue in a normal or near normal manner. These commands all use the FUNCTION key in combination with other keys. A complete list of all the commands and their meaning may be obtained by typing FUNCTION-HELP. The simplest and most useful is FUNCTION-ESCAPE, to be used when the message "Output Hold" appears in the bottom center of the screen.[12] More drastic measures are: FUNCTION-CONTROL-T and FUNCTION-CONTROL-CLEAR-INPUT.

### 17.2.4 Error Recovery

In rare circumstances, THOUGHTSTICKER may encounter an error. The user can often recover from these errors gracefully. A window will appear announcing the error and offering to give details which will appear if the user clicks the mouse in the window or types FUNCTION-0-S (the latter is prefered). The user is then given a range of options, one of which (usually chosen by the RESUME key) will restart the frame gracefully. Usually, THOUGHTSTICKER can continue from this point but the error may imply that the particular function that caused it cannot be used. However, the Symbolics attempts to offer alternate choices for recovery, and users are encouraged to read the choices in detail.

---

12. Under Release 4.5 of system software, the correct sequence is FUNCTION-TRIANGLE.

For example, if the system runs out of file space, the user will be offered an option to "expunge" the file system, thus creating more file space. After this is performed, THOUGHTSTICKER can be made to continue the process by another choice displayed with the error message, and without making the user repeat the command which caused the error.

# Chapter 18

## Acknowledgements

PANGARO Incorporated gratefully acknowledges all those who have contributed to the work herein described, especially Dr. Gordon Pask, for his originality in the expression of Conversation Theory and its knowledge calculus, Lp; Dr. Jeffrey Nicoll, responsible for the implementation, major portions of the user documentation, and who took on both the theoretical complexities of Lp and the unknowns of the early Symbolics environment; Heather Harney, for managing the creation of the user documentation, for contributing major portions of the user documentation, and for her support in the construction of the system; Amalya Dixon, for production assistance; Mr. Colin Sheppard, for his confidence and expert guidance as Contracts Officer in the project; Dr. Ken Gardner, Head of the Applied Psychology Unit of the Admiralty Marine Technology Establishment (UK) which funded the majority of the research; Mr Dik Gregory and Ms Rosemary Todd of the Applied Psychology Unit; Dr. Joseph Zeidner, former Chief, Dr. Edgar Johnson, Chief, and Dr. Robert Sasmor, Director of Basic Research, of the US Army Research Institute for their support. We are also grateful to the many researchers, some named and many more unnamed, of the days of System Research Ltd (UK), who spent countless hours in difficult conditions constructing THOUGHTSTICKERs, both digital and analog, electronic and not, since the 1950s.

- - end - -