A GUIDE TO TECTON AND NATURAL LOGIC

D. Kapur, D. R. Musser, A. A. Stepanov
General Electric Research & Development Center

1. Introduction

Tecton is the name for a formal approach to building system specifications and programs and for a computer language that embodies that approach. The essential concepts of the approach are based on a novel logical framework called natural logic. Both Tecton and natural logic are still under development; the purpose of this note is to give a very brief overview of the overall plans and guide the interested reader to several papers and working documents on both Tecton and natural logic.

The concepts of natural logic include naturalness of expression, emphasizing the ability to describe concepts, manipulate them, and reason about them at a very high level - because of, rather than in spite of, an underlying formal rigor that permits logical reasoning to be carried out with the aid of an automated inference system. A concomitant emphasis is on use of "common sense" modes of reasoning, permitting, for example, an appropriate handling of exceptional cases without introducing contradictions (similar to recent work on default reasoning systems).

Although work on Tecton and natural logic was originally motivated by applications in building and verifying complex software systems (see especially [1]), the approach and language have evolved to a point where they should be more generally applicable, especially in areas of artificial intelligence (AI) work such as knowledge representation, robotics, vision, and planning and problem solving. In fact, it would not be inaccurate to describe Tecton as an AI language.

A Tecton computing environment is being designed and will be implemented on Lisp Machines. Components of this environment will include an Editor, a Logical Analyzer, and a Librarian. The core of the system will be the Logical Analyzer, which will implement an interactive, automated inference system based on natural logic.

2. Tecton/1 and Tecton/0

A major goal of the design of the Tecton language is that specifications and programs should read almost like natural language (although this goal is different from work on natural language understanding systems). As part of the design effort we have developed a grammar for this natural language level of Tecton (ref. [3]), but we do not plan to implement this level immediately. Rather, we are first defining an intermediate version of the language that is syntactically simpler and more highly structured than natural language. This intermediate level is called Tecton/1, and will serve as a stepping stone to the eventual implementation of Tecton itself. Tecton/1 is currently being designed, and is described in the working document [6].

Tecton/1 is being defined with the aid of an even simpler core language, called the L language. L supplies the basic syntactic framework for Tecton/1, just as Lisp has done for many AI languages. This approach will also permit an experimental implementation of Tecton/1 to be accomplished fairly rapidly. (We already have implemented L on Lisp Machines.)

L, which might also be called Tecton/0, is described in Reference [8]. L is basically a subset of LISP, but its syntax is substantially richer. The basic approach to syntax has been taken from another LISP-like language, LOGO, but several extensions have been made. Semantically, L differs from LISP mainly in being somewhat simpler by elimination of all special functions (FEXPRs, LEXPRs, etc.). (LOGO retains a few special functions.) L has a richer set of conditional and iteration control constructs than in LOGO or many dialects of LISP, though not as extensive as those found in MACLISP and INTERLISP.

As an example of the difference between Tecton and Tecton/1, consider the following fragment of a Tecton specification of a distributed data base system:

> "There is a request that is not resolved"
> means there is a node X and a request R such
> that the status of R at X is neither
> "Accepted" nor "Rejected".
>
> "There is some message awaiting receipt
> in the system" means there is some
> node Y such that the queue of inputs
> at Y is nonempty.

This might be expressed in Tecton/1 as follows:

```
[there is a request that is not resolved]
  means
   [there is [.request .node]
     such that [status of .request at .node
                   is not "Accepted"
                and status of .request at .node
                   is not "Rejected"]]

[there is some message awaiting receipt
 in the system] means
  [there is [.node] such that
   [queue of inputs at .node]
     is not empty-queue]]
```

Syntactically, this would be acceptable already in L.

3. Natural logic

Natural logic is being built on two main cornerstones. One is a new approach to modal logic, i.e., the use of attributes attached to propositions, called modalities. Examples of modalities are: true, false, contrary, necessary, contingent, possible, impossible, provable, inconsistent, deterministic, absurd, meaningful, etc. In representing knowledge for reasoning about systems (including real world systems), many other modalities, such as default (normal), probable, plausible, desirable, interesting, etc., turn out to be useful. Reference [5] discusses a general framework for introducing modalities into logic; modalities are classfied based on their properties and connections to each other. The difference between our approach in [5] and traditional approaches to modal logic (such as are discussed in Hughes and Cresswell, An Introduction to Modal Logic) is that instead of concentrating on modal logic built around the pair of modalities (necessary, possible), we have tried to identify as many useful modalities as we can and build a natural classification of them. For example, we investigate the modality "normal," which captures a notion of default reasoning and is extremely useful in describing rules for dynamic hypothesis formation. The modal logic generated by this modality does not correspond to any standard modal theory (such as the S1 through S5 modal logics of C. L. Lewis).

The other cornerstone of natural logic is a "logic of objects." This logic deals with both real and conceptual objects. By real objects we mean objects in the real world (including, in some cases, objects in computer systems); examples are chairs, telephones, trucks, bank accounts, messages, transistors, people. By conceptual objects we mean collections of properties, such as a chair having four legs and being used for sitting, a telephone having a numeric address associated with it, and a checking

account having current balance and the necessity that the balance be positive. These are not definitions of real and conceptual objects; in fact we take these terms as primitive (undefined) terms, but attempt to give a framework in which it is possible to give a precise and rigorous definition of types of objects and of particular objects. We also want to be able to make effective use of properties of objects in logical reasoning about them - specifically for reasoning with the aid of computers. These issues are discussed at length in [4]. The discussion in the remainder of this note may however be useful in providing additional insight about the notions of real and conceptual objects.

Of course, most real objects cannot be "put in the computer"; they can only be represented by other (real) objects that are in the computer, called descriptors. Concepts we do not regard as being in the computer either. An individual property is represented in the computer by a real object called a proposition, while a concept consisting of a collection (often infinite) of properties is represented by a finite collection of propositions from which all properties belonging to the concept can be derived by logical reasoning.

Some additional examples may help to clarify the above remarks. Where, for example, do mathematical objects, such as numbers, matrices, rings, or sets, fit into the picture? We regard them all as concepts. Even the simple case of numbers is an illuminating example, since we often speak of numbers as being "in the computer." What is really meant, though, is that representations of numbers are in the computer. In our framework, numbers are concepts, and thus the representation in the computer is by propositions. The usual kind of representation, as a sequence of bits, is not what we would ordinarily think of as a proposition; but our notion of proposition encompasses it. What we actually have is that the string of bits is not by itself the concept of the number, but is part of a collection of propositions that together define the number. The other parts of this collection would be propositions about the hardware or software in the computer that provide an interpretation of the bit sequence as a number. (References [2] and [7] contain additional discussion of the role of mathematical objects in Tecton.)

Secondly, how do we "really" deal with real objects, such as trucks and people, when all can have in the computer are descriptors? Part of the answer is that we cannot deal with them completely; we are limited in essence to some form of "pointing" to real objects, by means of real objects that we can hold in the computer. But this pointing should not be limited to being done just with descriptors, though it has been in almost all computer languages, including database systems. By using concepts - or actually the propositions from which the concepts can be logically derived - we can greatly enrich our capability to be precise in pointing to real objects and describing the relations between them. Computer systems are almost always used today in a

way in which the role of concepts is suppressed - it is there, but it is in the background as a lot of implicit and unorganized assumptions. (We believe that this lack of emphasis on concepts and reasoning is to a large degree responsible for the excessive complexity, unreliability, or unsuitability of computer systems themselves and of the applications that are developed on them.)

Another question that might be asked is whether descriptors are the only real objects that can be in the computer. According to the view we take, the answer is no; a text editor buffer is an example of an object whose purpose is not to describe something else, but to be part of a mechanism that is useful in itself.

4. References

1. D. Kapur, D.R. Musser, and A.A. Stepanov, "Tecton: A Language for Manipulating Generic Objects," Proceedings of Program Specification Workshop, University of Aarhus, Denmark, August 1981, Lecture Notes in Computer Science, Springer-Verlag, Vol. 134, 1982.

2. D. Kapur, D.R. Musser, and A.A. Stepanov, "Operators and Alge- braic Structures," Proceedings of the Conference on Functional Programming Languages and Computer Architecture, Portsmouth, New Hampshire, October 1981.

3. D. Kapur, D.R. Musser, and A.A. Stepanov, "Syntax of Tecton," Working Notes, GE Research & Development Center, February, 1983.

4. D. Kapur, D.R. Musser, and A.A. Stepanov, "Notes on a Logic of Objects," Working Notes, GE Research & Development Center, July, 1983.

5. D. Kapur, D.R. Musser, and A.A. Stepanov, "Modalities, Abstraction and Reasoning," Working Notes, GE Research & Development Center, July, 1983.

6. D. Kapur, D.R. Musser, and A.A. Stepanov, "Notes on the Tecton/1 Language," Working Notes, GE Research & Development Center, July, 1983.

7. D. R. Musser and D. Kapur, "Rewrite Rules and Abstract Data Type Analysis," Proceedings of Computer Algebra: EUROCAM '82, ed. by J. Calmet, Lecture Notes in Computer Science, Springer-Verlag, Vol 144, April 1982, pp. 77-90.

8. D. R. Musser, "The L Programming Language," Working Document, GE Research and Development Center, May, 1983.