Memoria de la

CONFERENCIA INTERNACIONAL SOBRE SISTEMAS, REDES Y COMPUTADORAS. MEXICO 1971

Centro Vacacional del I.M.S.S.
Oaxtepec, Morelos, MEXICO
19 a 21 de enero de 1971

VOLUMEN II

MEMORIAS

CONFERENCIA INTERNACIONAL IEEE MEXICO 1971

SOBRE SISTEMAS, REDES Y COMPUTADORAS

OAXTEPEC, MOR., MEXICO

ENERO 19 - 21, 1971

EDITADO POR:

A. Alonso Concheiro
R. Canales Ruiz
L. Rodríguez Viqueira

AN APPROACH FOR MAPPING ABSTRACT INFORMATION STRUCTURES ON DIGITAL COMPUTER MEMORIES

Carlos J. P. Lucena
Departamento de Informática
Pontifícia Universidade Católica
Rio de Janeiro - Brasil

Summary

One of the strongest motivations for research in the area of Computer Science is to give software a firm basis. By firm basis we mean a systematization of concepts, but not unecessary formalism.

The first issue that arises when one explores along these lines is the need for a unified principle for computer information storage. The mapping of abstract or concrete models into storage should ideally be performed in a well defined fashion.

In our work we propose one theory for representing information structures in computers. Instead of proposing a "general recipe" for data structures, we design a formalism that enables a constructive study of modeling techniques in programming. The system emcompasses all the know types of data structures. The theory of directed graphs is used as the basic tool for this work.

An attempt is made to generalize graphs to structures involving vertices (nodes)composed of various information sets with connections among them.

The basic element of our system is called an "information module". It contains features which establish interconnections with other modules (variable number of links), as well as descriptors that describe the information associated with the module. A descriptor can be a digraph in which each vertex (module) can also be a digraph.

Some interesting results can be obtained when we design algorithms in the context. The work also discusses some issues related to the implementation of the proposed system.

General Concepts

The abstract representation of a problem on a digital computer may be graphically depicted by the following diagram:

$$S_i \rightarrow \boxed{\quad P \quad} \rightarrow S_\sigma$$

where $S_i=\{c_1, c_2, \ldots, c_n\}$ and $S_\sigma=\{c'_1, c'_2, \ldots, c'_n\}$ are **strings** of input and output symbols, respectively.The box represents a processor containing P, a plex. S_σ will be empty in the case of unsolvable problems.

A plex [1] is a complete model of a concrete or abstract entity composed of:

a)- algorithms
b)- structure

We must now define the meaning of the plex's components. An **abstract alphabet** is any finite set of objects called letters. A word in an abstract alphabet is any finite and ordered sequence of letters. An alphabetic operator or alphabetic mapping is any function that establishes a correspondence between words of an alphabet (A_i) and words in the same or in another alphabet (A_σ). An algorithm [2] is a function contructively specified between words of abstract alphabets.

The process of specifying an algorithm is accomplished through algorithmic systems. These systems include entities called elementary operators and elementery discriminators. The elementary operators are simple mappings from A_i to A_σ. A sequential execution of them specifies an algorithm.

The discriminators detect the existence of certain properties in the information being processed, changing the sequence of the elementary operators accordingly. In programming, the specification of an algorithm is generally done through a directed graph called a flowchart. In programmed algorithms $A_i \equiv S_i$ and $A_\sigma \equiv S_\sigma$.

Information units do not appear as a mass of unordered items. The organization of the words of the input alphabet in the active zone (memory) of the processor is called the structure of the information units. The form is which they are allocated in memory influences the process of ordering the elementary operators and discriminators for the composition of an algorithm. Formally, we call an information structure a directed multigraph M_g, defined by the ordered 4-tuple $<N, L, P, n_0>$, where:

1. N is a finite non-empty set of elements called vertices (nodes) of M_g. To each node $n \in N$ there is an associated range Dw, which is a set of computer words.
2. L is a non-empty set of elements called labels of M_g.
3. P is a non-empty set of ordered triples $(n, 1, n') \in N \times L \times N$ called arcs (pointers) of M_g.
4. $n_0 \in N$ is a distinct node, called the departure point for an application

of the structures.

A finite path of M_g is a finite sequence of k arcs, k>1, of M_g of the form:

$$n_{i1} \xrightarrow{l_{i1}} n_{i2} \xrightarrow{l_{i2}} n_{i3} \cdots n_{ik} \xrightarrow{l_{ik}} n_{ik+1}$$

Note that the definition allows for arcs with the same direction but different labels to connect the same two vertices. The existence of different kinds of relationships between two vertices makes M_g a multigraph instead of simply a graph [3].

What will be discussed next is an approach to systematically building an information structure [4] in a digital computer, as an attempt to suggest efficient modeling techniques in programming.

Definitions: The Modular Structure

The structural space of a program is an entity over which will be applied a programmed algorithm. This entity is defined as a 4-tuple S = <M, I, F, Ls> where:

1. M is a mapping of an information structure into the storage unit of a digital computer, called a modular structure.
2. I is a set of words in a computer memory that stores the strings of input symbols S_i.
3. F is a set of formats that describes the nature (type) of the words in I.
4. Ls is a list of available space (containg nodes of variable sizes).

Note that I contains the information units, M describes the structural relations among the words of the input alphabet. F and L are two technical impositions dictated by the design of existing digital computers.

In accordance with the definition of information structure, M is a 4-tuple <M*, L*, P*, R> where the components of the definition mean:

1. M* is a non-empty set of information nuclei (defined later).
2. L* is a non-empty set of labels (integer constants) associated with each nucleus.
3. P* is a set of pointers that interconnect the several nuclei N of the structure.
4. R is a pointer that indicates the initial nucleus for an application.

In practice, an element of the structure is a module m defined as a pair (n,e) ε M* × P*. The nucleus n represents the range Dw associated with each vertex of M_g and is physically composed of three pointers; e represents all the pointers that originate from a module. Graphically the structure may be represented as in figure 1.

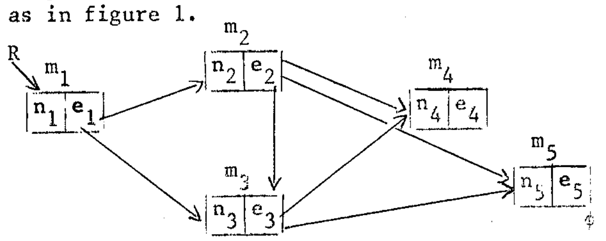

Figure 1

In figure 1 ϕ represents a set of null pointers. A module m can be better seen through the illustration in figure 2:

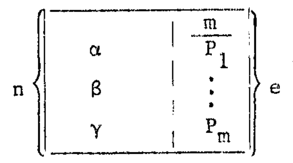

Figure 2

The dotted linke in figure 2 has a meaning that will be explained later.

The pointers {α,β,γ} that comprise n have the following goal:

1. $\alpha=\phi$, if m is a terminal module; if the module is a non-terminal one, α points to the origin R os the inner structure.
2. $\beta=\phi$, if m is a non-terminal module; if the module is a terminal one, β points to the beginning of the area A which contains the information associated with the module.
3. $\gamma=\phi$, if m is a non-terminal module; if the module a terminal one, points to the format in F that describes the information associated with m.

The elements m, P_1, ... , P_m of e are the number m of connectors P_1, ... , P_m originating from a nucleus.

The concept of terminal and non-terminal modules allows the information associated with each module to be decomposed and described by inner structures. Although the concept of a graph is independent of dimension in the euclidean sense, the pictorial representation in figure 3 makes the last explanation more explicit.

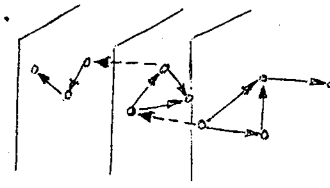

Figure 3

It is easy to show that the proposed structure emcompasses the current particular models of information strustures [5] e.g.: trees lists, rings, etc.

Two graphical illustrations of this fact follow (figure 4)

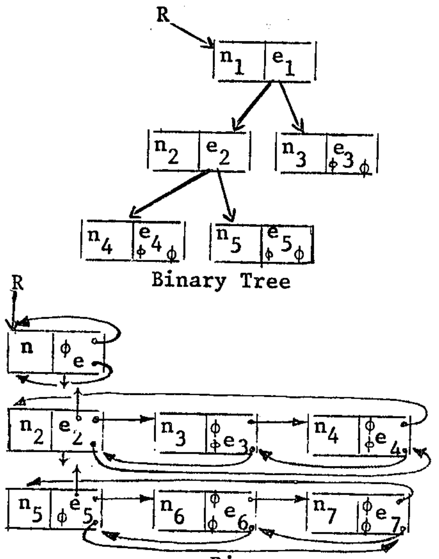
R
n_1 e_1
n_2 e_2
n_3 e_3 ϕ ϕ
n_4 e_4 ϕ ϕ
n_5 e_5 ϕ ϕ
Binary Tree
R
n ϕ e
n_2 e_2
n_3 ϕ e_3
n_4 ϕ e_4
n_5 ϕ e_5
n_6 ϕ ϕ e_6
n_7 ϕ ϕ e_7
Ring

Figure 4

This fact is due to the ordering of the pointers in e.

During the modeling process of the solution to a problem, the concept of inner structure permits elements with the same degree of detail to be represented at the same level. For example, in a compiler construction problem the first level may be a set of interconnected procedures, the second a set of interconnected statements etc; in a psychological application the first level may be a set of interconnected families, the second a set showing the connections among the eldest sons of each family etc.

Types of Information and its Formats

The area of information relative to each module is pointed by the pointer β of a terminal module. The content of the memory area associated with the module is described in the format indicated by γ. The format describes the characteristics not only of the informations associated with the vertices (M or M*) but also with the arcs (P or P*)of the structure(if any).

Each element of the set F is a format with essentially the same characteristics of a symbol table. It contains the names, types and dimensions of the variables associated with the module.

An important characteristic of the internal representation is the variety of information types provided by the system. Besides the conventional types like real, character etc, the system requires the types FORMAL and DYNAMIC. The DYNAMIC allows the declaration of variables as push-down stacks and the type FORMAL declares that a variable may assume a formula as its value.

In the second case one may have, for instance, a logical expression whose result may be obtained from values of previously defined logical variables. The values may be tested later by the program that manipulates the structure. This last feature allows, for example, the implementation of models in the theory of programming such as interpreted graphs. For that we would have variables of the type FORMAL associated with the pointers of the structure[6,7].

Internal Organization of the System

We shall now make a few comments about some practical aspects of the implementation of the system. The reason for the dotted line in figure 2 is that n and e will be considered different nodes in Ls, the list of available space. Ls will contain nodes of different sizes The list will grow from the two extremities H_1 and H_2 (bottom up and top-down) with a control to avoid collision. A different garbage collection strategy is provided for each direction.

A linked storage scheme is provided for m. A last important fact to be mentioned is that a strategy of dynamic storage allocation is used for the management of the area I. The scheme in figure 5 resumes what was said about the internal mapping of the described system.

These strategies are required in order to perform operation on structures: addition or deletion of modules, concatenation of structures etc.

Applications

In order for the proposed system to operate, procedures are required for:

a)- Modular structure construction
b)- Format specification
c)- Structure traversing
d)- Modular operations such as concatenation, insertion and elimination

It is extremely desirable that programs utilizing the proposed model be simple. There is a measure for this simplicity.

A program written in a higher level language which utilizes the conventional structures, e.g. arrays, has the same effect as a program operating over a structural space of just one module.

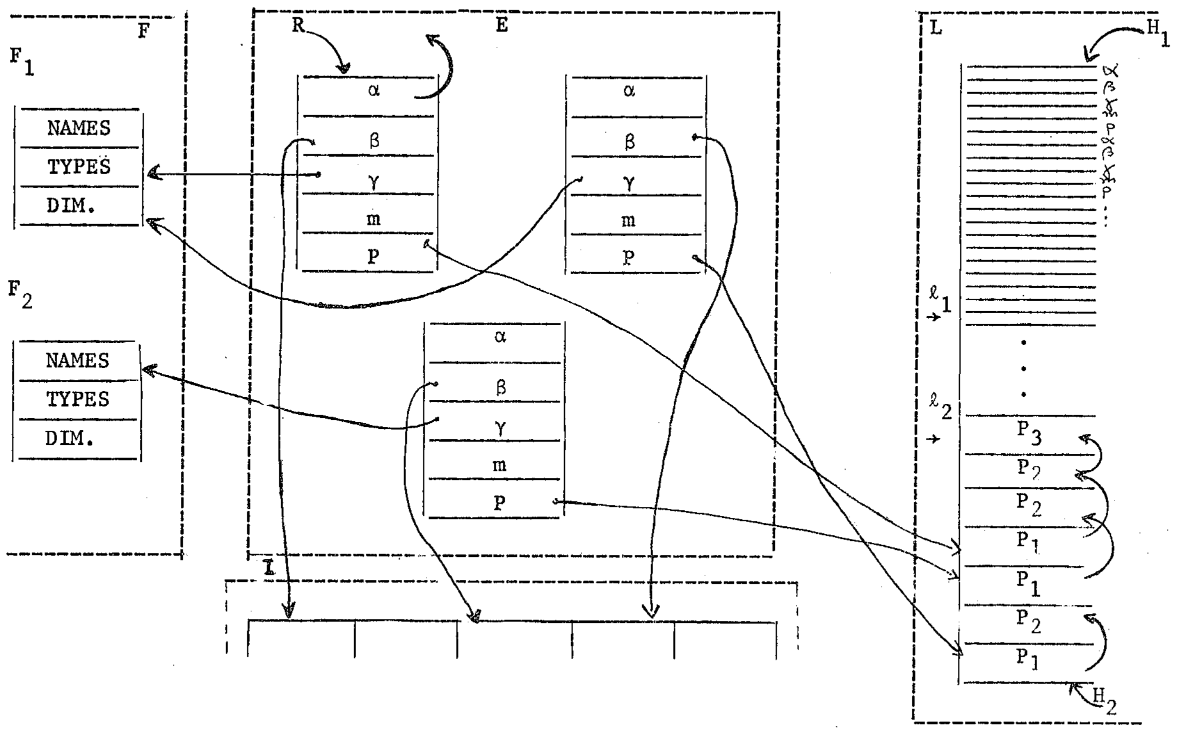

Figure 5

All the extra apparatus required for processing a single module compared to the conventional program defines the degree of complexity of the system.

In our implementation we opted for embedding rather than the creation of a new language.

Here the problem is to decide whether to use special procedures or to alter syntax of the chosen language, keeping in mind that we want to preserve the language's degree of proceduralness [8].

Since the storage scheme is of general use we list some diverse applications: linguistics, information retrieval, automata theory, compiler construction, network theory and structural interrelations in the social sciences.

References

1. Ross, D. T. - "The AED Free Storage Package" CACM 10,8 - Aug. 1967.
2. Glushkov, V. M. - "Introduction to Cybernetics" - Academic Press - 1966.
3. Berge, C. - "Théorie des Graphes et ses Applications" - Dunod - 1958.
4. Evans, D. - "Data Structure Programming System" - Proc. IFIP Conference - 1968.
5. Knuth, D. E. - "The Art of Computer Programming" - Vol. 1 Addison Wesley, 1968
6. Floyd, R. W. - "Assigning Meaning to Programs" - Proc. AMS Symposia in Applied Mathematics - Vol. 19.
7. Manna, Z. - "Termination of Programs Represented as Interpreted Graphs" Proc. AFIPS - Spring Joint Computer Conference - 1970.
8. Sammet, J. E. - "Programming Languages: History and Fundamentals"-Prentice Hall 1969.