NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

CCP-254 dmj

68-10 cop. 2

CABAL - ENVIRONMENTAL DESIGN OF A COMPILER-COMPILER

Richard K. Dove

Carnegie-Mellon University
Pittsburgh, Pennsylvania
May 12, 1968

Address: Carnegie-Mellon University
Computation Center
Pittsburgh, Pennsylvania 15213

Phone: 683-7000, Extension 268

ABSTRACT

CABAL is a compiler-compiler designed at Carnegie-Mellon University. In appearance its debt to ALGOL, PL/I, and FSL is apparent. Additionally, it features a formal co-routine structure; a unitized data structure which can function as an array, scalar, list, tree, stack, queue and plex; and a high level code generation facility. In addition to presenting these and other results of the CABAL design, the design process itself is examined. Primarily, the results of the design process are viewed as owing their existence to a hard look at defining the ultimate environment in which the system would function. Environmental analysis and goal definition are separated from the design process and held accountable for the result.

This project was funded by ARPA.

I. INTRODUCTION

CABAL[1], a compiler-compiler design project at Carnegie-Mellon University, was fostered in the wake of Jerry Feldman's FSL thesis[2]. Initial plans were to use Feldman's basic Production Language - Formal Semantic Language structure, include extensive storage allocation procedures, and implement a number of extensions suggested by feedback from an FSL implementation of FORMULA ALGOL[3,4].

As the project progressed a study of the current state of the art for both translator writing systems and language design methods gradually changed the design goals of the development group. Working papers published throughout the design reflected this continuous change. Eventually a well-defined set of goals evolved and the design phase progressed to a solution not closely resembling the original direction.

It is an accepted fact that large design projects very rarely end up as they are initially conceived. A learning process goes on when the problem is seriously attacked which precludes a static set of goals. However, in the course of checking the pertinent literature and at the same time noting the causes for progressive quantum jumps in the CABAL design effort, two things became apparent: first, as a strong set of goals developed the design solutions came rapidly; and second, the literature notably lacks in design methods as well as goal definitions and language design considerations.

Shifting now to the results of compiler-compiler or translator writing system design, the current literature reflects the youthfulness of the field. New concepts and solutions to new problems affect systems design in a predictable way: the primary concern is to demonstrate a working system.

Only after we relegate these new problems to an innocuous position and comfortably understand the new concepts involved do we turn our attention to usable systems as opposed to workable systems.

Although compiler-compiler techniques are by no means near to being a closed issue, there is enough foundation material currently available that a little bit of polish can produce a palatable system.

Thus, the concern of this paper is twofold. First, in an effort to increase documentation of design processes as well as stimulate some criticism toward better methods, I have included a section covering the goals set forth for CABAL as well as the information responsible for setting them. The point of view was stimulated by something T. E. Cheatham lightly dropped: "We are concerned hardly at all with the extremely important and often neglected problems of the environment in which a compiler or code resulting from a compiler is to operate."*

The second aim of this paper is to present a substantial portion of the CABAL design results. Briefly, the language owes much of its appearance to ALGOL, PL/I, and FSL. Additionally, it provides a formal co-routine[6] structure syntactically patterned after ALGOL procedures, and a unitized data structure capable of functioning as a scalar, array, list, stack, queue, tree and plex. There is a clean separation between syntax, semantics, and code generation facilities without sacrificing a unified language. Lacking, however, is a discussion of general purpose input/output facilities as their inclusion at this point is not considered enlightening.

The remainder of this paper is sequenced in five sections followed by an evaluation of how well the design satisfied the goals. After discussing

*Cheatham, T. E.[5], p.65

design criteria in the next section, the language is presented as structure and control flow, semantics processing, syntax parsing, and code generation. Although this breakdown sounds like a delivery of semantics language, syntax language, and code generation language, the individual language distinction has been carefully bred out of them. The presentation sequence was chosen for how much light each shed on its followers.

II. DESIGN CRITERIA

CABAL Note 3[7] presented the initial thoughts of the CABAL Development Group on design criteria for compiler-compilers in general. The following discussion presents the criteria governing CABAL specifically and includes pertinent material from Note 3 as well as additional constraints determined since then.

Much of the resultant criteria is strongly based on my evaluation of where Carnegie's computation center and computing in general is going in the next five to ten years. Additionally, the fact that we are "going" is most important. The truth of change must be recognized if anything lasting is to be built.

A Data Base

Rather than arbitrarily choosing guidelines which sound intelligent and meaningful, we should first examine the ultimate environment in which CABAL will function. In fact, there are three environments of concern: physical system, user, and maintenance. In addition, both present and future definitions of these environments must be investigated, else another obsolete system be designed.

The physical system environment contains both hardware and software, the things CABAL must directly interact with. The overall environment I see as a computer utility, already started and surely on its way to great expansion. This means a hardware configuration consisting of many different kinds of remote terminals as well as a complex processing facility. This processing facility contains a multiple number of central processors of unlike design, some will be interconnected and some not. Additionally, parallel processing is expected.

The software environment is felt on two levels. Operating systems capable of time sharing, batch processing, and parallel processing, will buffer all interactions between the processing facility and running programs. On another level we find that many different kinds of languages are present ranging from assemblers and compilers to interpreters as well as from general to special purpose.

Academic surroundings make the maintenance environment different from its commercial and industrial counterparts. At least, defining the maintenance environment with the least amount of restrictions on it proves such. The academic computation center has a high turnover of employees due to student labor and lure of better wages in industry. Additionally, many employees, barring permanent staff, are not overly endowed with systems experience (maintenance procedures or systems comprehension). Then, too, we seem to have more systems than maintenance personnel. Thus, inexperienced personnel are used for much of our maintenance. Teaching them complex systems and techniques takes so much time that few of them have the chance to become indispensable. Only in this last respect is our maintenance environment enviable.

The user environment is also unique to an academic atmosphere. Researchers working on compiler-compiler techniques, languages, and large programming systems are only outnumbered as a group by the mongol hordes from the undergraduate ranks. Then we have the graduate students whose theses involve special purpose languages to implement their main concern. Finally, there is enough on-site talent looking for means to express itself that lowering the amount of pain incurred to implement a system will considerably increase the amount of talent used.

So far we have only been concerned with environmental factors. I have tried not to introduce any artificial constraints which would result by drawing conclusions from these factors. Separating the one from the other clearly defines a base for design criteria. When considering change it will be easier to evaluate what is being given up against what is being gained without confusing the issue with secondary effects.

General Design Goals

Directing our attention now to the design goals of CABAL, we can separate our concern into three areas: syntax parsing, semantic processing, and code generation. Although CABAL as a whole should exhibit unity, the problems associated with each of these three areas are distinct and therefore warrant individual consideration. Additionally, one more area of concern germane to programming languages in general, structure/control flow considerations, is readily isolated. However, before considering each of these four areas individually, I will outline a list of unifying overall design goals and justify each by its relevance to our data base.

Modularity is important. In designing a system one seeks to divide it into distinct sub-sections. The criteria governing this separation is to produce sub-sections which have a minimum of interaction and inter-communication and a maximum of structural similarity. The resultant modularity offers a system which can be easily changed, debugged, and extended; has fewer initial bugs; is easily and quickly tested; is quickly understood by design and maintenance personnel; is less confusing to users; and possesses an inherent division of labor for both the initial coding and subsequent maintenance. Then, too, if this modularity shows through at the user level, it becomes a system whose use is quickly learned, readily understood, and easily retained[8].

The language must be readable and conceptually simple, facilitating a quick grasp by the occasional or one shot user as well as providing for comprehension by maintenance personnel and programmers faced with the problem of understanding someone else's program. For the same reasons interactions with system parts concealed below the surface must be kept to a minimum. At the same time the language should be sufficiently powerful and terse to warm the heart of the most esoteric systems programmer, providing him with handles on any part of the system.

Flexibility must be inherent as the environment will most definitely change. Machine dependence should be kept to a minimum as should well-defined interactions with the system environment. Note that modularity does much for flexibility but does not insure it.

Most important is applicability. CABAL is useless unless its performs some meaningful unique function geared to the problem at hand: compiling compilers. Thus, the language must apply to the problem area.

Compatability with the physical system environment must be insured. Reflecting the physical system in CABAL buys power from another angle. Not only should CABAL fit into its environment, but it must provide access to all parts that are usable.

Its range must be comprehensive enough to warrant its development. The ability to produce conversational and incremental compilers should be considered. It should be able to compile assemblers and interpreters as well as compilers. One benchmark would be the ability to compile itself.

Last but by no means least is reliability. This aspect has been discussed at length by Peter Naur[9]. I have saved mentioning reliability until this point as many of the preceding goals do much to insure it. However, other goals aside, reliability must be recognized as a primary goal in and of itself.

Collecting these goals into one concise list we find the following:

1. applicability to compiling language translators;
2. wide range of producible translators;
3. compatability with total dynamic environment;
4. modularity;
5. flexibility in dynamic environment;
6. power for the professional systems programmer;
7. transparency for the non-professional user;
8. readability;
9. terseness for the professional user;
10. machine independence;
11. reliability.

This list is not meant to be composed of first principles and therefore we need not be concerned about overlap. In the following section we will see how these goals have affected the design of CABAL. The success of the design effort is measured by how well it satisfies these goals. Consequently, the formation of design goals is seen to be of prime importance, for the design effort is determined by them.

III. STRUCTURE AND CONTROL FLOW

In order to avoid confusion among the various levels of processors and languages associated with a compiler-compiling system, I will adopt a convention defined in CABAL Note 3. C^2, C^1, and C^0 will refer respectively to the meta-compiler CABAL, an object compiler written in CABAL, and an object program written in the language of the object compiler. When necessary, subscripts P and L will further distinguish processors from languages. "Thus, C^1_P is a processor written in C^2_L, C^0_P is a program written in C^1_L, and C^0_L might be considered the 'language' in which data for C^0_P is written."*

In this section we are concerned with structure and control flow associated with C^2_L. Structure and control at lower levels will be discussed in the sections concerned with code generation.

Structure

CABAL is an ALGOL-like language and in fact has ALGOL's block structure complete with BEGIN-END pairs and identifier scope. In addition, the internal structure of a block is divided into a declaration part followed by a statement sequence part. Among other things, the declaration part includes declarations for two programming structures: co-routines and ALGOL procedures. A co-routine declaration is syntactically similar to an ALGOL procedure declaration and will be discussed shortly.

*Shaw, M. and Fierst, J.[7], p. 19.

Another kind of structure inherent in CABAL results from the distinction between syntax parsing, semantics processing and code generation. The next three sections will discuss each of these at length so at this point I wish only to indicate the nature of their separation. Syntax parsing is provided by reduction statements modeled after those used by FSL. However, they are not format dependent in keeping with the ALGOL-like nature of CABAL. Code generation uses FSL's code bracket concept but the contents are entirely different. Finally, semantics processing is provided by constructs similar in number and nature to ALGOL statements.

At a lower level modular concepts are applied to the structure of the various language constructs. Concepts with similar meanings exhibit similar structure. For example, the syntax for accessing an element within a multiple element data structure uses square brackets around the indexing information. An extension of this concept demanded square brackets around the indexing information associated with accessing bit fields within elements. Thus, A [3].[4] requests bit number 4 of element number 3 in data structure A.

Applying modular structural concepts again resulted in declarations resembling those of PL/I rather than ALGOL's. Whereas ALGOL has individual declarations for each kind of typed storage structure, PL/I has a single DECLARE construct which is parameterized. Extensions can easily be accommodated without adding a new declaration concept.

Modular structure concepts and simplicity considerations were responsible for unitizing five data structures into a single concept. An examination of data structure declarations will finish the structural concepts. CABAL has only one data structure. However, it has the capability to function as a scalar, array, threaded list, stack, or queue. The syntax is:

```
<declaration> ::= DECLARE <declaration list>
<declaration list> ::= <name list> ( <structure> )
        | <declaration list> , <name list> ( <structure> )
<structure> ::= <nature> <element linkage>
<nature> ::= NUMBER , <bound> <number option>
        | Logic , <bound>
        | STRING , <bound>
        | NAME <bound>
<bound> ::= <number expression> | ? | ? <number expression>
<number option> ::= , FIXED | <empty>
<element linkage> ::= , <bound pair list> <pop linkage>
        | <empty>
<pop linkage> ::= , LIFO | , FIFO | , RANDOM | <empty>
<bound pair list> ::= <bound> : <bound>
        | <bound pair list> , <bound> : <bound>
```

Examples follow for scalars A and B, array C, stack D, and list E:

```
DECLARE A,B(NUMBER,6),
        C(LOGIC,4,[0:10,0:10]),
        D(STRING,10,[1:?15],LIFO),
        E(STRING,?,[1:?],RANDOM);
```

Thus, if there is no <element linkage> a scalar is signified, <element linkage> without <pop linkage> results in an array, and the presence of <pop linkage> signifies how elements will be pushed and popped for stacks, queues, and lists. Plexes and trees can be built from lists.

The expressions following the type indicate how many significant figures associated with the type in question are to be retained. Limits on these will be set by the implementation. A ? signifies an unknown number and results in dynamic allocation. An expression following a ? indicates the user's guess and allows the compiler to take advantage of this information for efficient structuring.

CABAL has only four types: NUMBER, LOGIC, STRING, and NAME. These will be discussed in the semantics processing section.

Control Flow

Control flow is altered with GOTO and call statements. The GOTO statement, in addition to its standard GOTO <label identifier> appearance, also appears as GOTO <name variable>; where <name variable> can be any type NAME data structure element with a LABEL value. Thus, the ALGOL switch label construct looks like GOTO A[N] where A is any multiple element data structure of type LABEL and A[N] has a LABEL value.

There are three kinds of call statements, however, each is syntactically identical. The difference is in the item being called which can be function designators, procedures, and co-routines. The method of calling is standard ALGOL in nature: <identifier> (<parameter list>).

Syntactically co-routines are identical to procedures with the exception of a COMMON declaration which may appear in a co-routine but has no meaning in a procedure. Briefly, co-routines are programming structures which, when they receive control, continue processing from the internal point which last gave up control. Thus, although control is passed to them by name, they can be considered as having multiple entry points. The COMMON declaration is simply a group of statements which are executed everytime control enters the co-routine in concern. Immediately after execution of the COMMON statement control passes to the internal point which last gave up control.

Calling a co-routine demands that control be returned to the point following the call, just as with procedure and function designators. However, co-routines may also be activated by GOTO statements. When this is the case, a co-routine must relinquish control with a GOTO statement rather than a RETURN as could be the case had it been called.

Minor control changes associated with IF and FOR statements are similar to ALGOL and need no explanation. The following section on general semantics processing will present their exact nature. Reduction statements exhibit control changes similar to IF statements and will be further explained in the section concerned with syntax processing.

IV. GENERAL SEMANTICS PROCESSING

T. E. Cheatham has noted that - "While there exist reasonable elegant schemes for 'automatically' doing syntactic analysis (and even much of code synthesis), the handling of declarations is generally messy with any but the simplest of languages. For this reason (among others) it will prove highly useful in any general purpose compiling system to have the ability to do arithmetic and relationals - i.e., the 'action language' should contain at least the rudiments of a good algebraic language."*

Rather than just containing "rudiments", CABAL has all the power of ALGOL as well as some significant extensions, notably co-routines, stacks, field and bit level addressing within storage elements, and the reduction statement which is not limited to syntax parsing alone.

*Cheatham, T. E.[10], p. 65.

Expressions and Types

Expressions have four types as do data structures. However, typing in CABAL does not follow the usual rules. An expression type is determined purely by the operators appearing in it unless it has only a single operand, in which case it takes on the operands type. Thus, a string variable could be numerically added to a logic variable and would produce a value of type NUMBER. The main reason for typing variables is so that information within them may be partially accessed. For example: <variable>.[<field designator>] is a means to address a portion of the variable. If the construction was A.[2:4] we would be addressing the second, third, and fourth digits within element A where digit is defined according to the type of A. Digit definitions are as follows: NUMBER-decimal digit, STRING-alphanumeric character, LOGIC-one bit, NAME-one entire name.

The unary and binary operators are classified according to the expression type they produce.

<unary numerical operator> ::= - | + | ↓

<binary numerical operator> ::= - | + | / | * | ↑ , o , C

<unary logical operator> ::= ¬

<binary logical operator> ::= < | > | ≥ | ≤ | = | ≠ | ∧ | ∨

<binary string operator> ::= &

Some of the unfamiliar operators are &:concatenation,o:mod, C: is a substring of, and ↓:truncation. The substring operator provides a value of 0 if the left string is not contained in the right string and a value corresponding to the digit position of the left string's first character match in the right string if it is contained.

Name expresssions never contain operators as their function is to convey pointers. For example, a name variable N might be assigned the name of a label, N←NAME(LABELX), and then appear in a subsequent GOTO statement. Name variables may be used in place of any CABAL name provided they have been assigned either the name, copy, or form of the item they are representing. Assigning N[X]←A allows N[X] to be used in place of A as N[X] has a pointer to A. However, if the value of A is changed then the value of N[X] has also been changed. To circumvent this, N[X]←COPY(A) will duplicate A and assign a pointer of the copy to N[X]. Additionally, N[X]←FORM(A) will allow N[X] to be referenced as if it had been declared the same way A had. The name variable concept is particularly useful in building up threaded lists or trees.

Statements

There are four statements specifically supplied for semantics processing: assignment, conditional, iterative, and push-pop. The assignment statement is straight forward:

```
<assignment statement> ::= <variable>←<expression>
<expression> ::= <name expression> | <number expression>
        |<logic expression> | <string expression>
```

The replacement operator does not affect type as the value of the <expression> is stored in the variable with no regard to type differences. If the length of the expression value cannot fit in the variable, then the replacement is undefined.

The conditional is standard ALGOL and needs no further explanation. The iterative statement is as follows:

```
<iterative statement> ::= <iterative condition> DO
<iterative condition> ::= WHILE <logic expression>
        | FOR <variable> ←<number expression> <terminal condition>
<terminal condition> ::= TO <number expression> BY <number expression>
        | TO <number expression> WHILE <logic expression>
        | WHILE <logic expression>
```

Although the syntax has been shortened, the semantics here are similar to ALGOL for statements.

The push-pop statement is used to push and pop elements associated withLIFO, FIFO, and RANDOM linked storage.

```
<push-pop statement> ::= <stack operator> <name>
         | <stack operator> <group reference>
<stack operator> ::= Δ | <stack operator> Δ
         | ∇ | <stack operator> ∇
<group reference> ::= <name> [ <group definition> ]
<group definition> ::= <index range list> <number expression list>
<index range list> ::= <empty> | * , | <index range list> * ,
```

To exemplify the range of manipulation this buys consider the following structure:

```
DECLARE MST(LOGIC,32,[1:X,1:Y,1:Z],RANDOM);
```

This gives us a master symbol table similar to the one which must be declared by the user at a block level sufficiently high to encompass all code bracket statements. Z is the maximum element number in the list and corresponds to the maximum number of identifiers the user's compiler will handle; Y specifies the number of 32 bit storage elements needed per identifier; and X specifies the number of remembered nested declarations any one identifier may have. By asking for RANDOM linkage we can push and pop any group of elements in the list at any time. The result of pushing element [*,*,2] is to have its address now by [*,*,3]; [*,2,2] goes to [*,3,2]; and [2,2,2] goes to [3,2,2]. Popping any one of these, though, removes it entirely from the list and changes group [*,*,3] to [*,*2] and so on.

Consequently, using this as our master symbol table allows us to push and pop information concerning every declaration of a given identifier name as well as only that information associated with its declaration at a specific block level.

The master symbol table (MST) serves as a communication link between the semantics processing and code generation. As will be discussed in the code generation section, storage allocation and other necessary MST information is entered into the MST by the code generators. During semantics processing this information as well as additional data the semantics routines may store in the MST is available for use.

V. SYNTAX PARSING

Syntax parsing is accomplished with a reduction statement similar to FSL's production except the stack manipulation is different. An example will clarify:

```
L:| "WHILE",EXP | → STA | WHCODE,RETURN |;
```

The sequence of events is as if an IF-THEN statement were executed. If the top stack entries are matched with the left section then the right section receives control, just before the right section gives up control totally the top two stack elements are popped and the contents of the middle is pushed on. Thus, although WHCODE is the semantics routine procedure called for processing, the stack remains the same until just prior to the RETURN execution.

Another difference from FSL is that there is not a semantics stack automatically pushed and popped as the syntax stack is. The user must provide his own semantics sequencing. This eliminates a lot of unnecessary stack manipulation as well as the confusion which arises in the following:

```
|A,A,B|→A,B||;
```

The question as to which A semantics should be retained does not occur without a parallel semantics stack.

Elements in the left and middle sections of the reduction statement are of two kinds: literals and meta-characters. The literals cause straight forward comparisons between the character string enclosed in quotes and the character string associated with the corresponding stack position. Meta-characters represent a group of literals and are matched if the corresponding stack position matches any one of the group.

CABAL associates a meta-character with its member literals through a define declaration. An example covering unary and binary numerical operators follows:

```
DEFINE UNOP="-"|"+",
           BNOP=UNOP|"/"|"*"|"↑"|"o"|"C";
```

Additionally, meta-characters can be defined for non-printing characters like end-of record, end-of-file, carriage return, tab, space, backspace, etc. by using the numerical equivalent of the character in question as a definens.

```
DEFINE EOC = 64;
```

The matter of the sigma function, a meta-character which always matches, is taken care of by the absence of either a meta-character or literal, when one is clearly called for by the comma placement. Alternatively, for those who demand a printable sigma function, a meta-character may be used whose definens is empty.

Actually, reductions are not restricted to operating on stacks, any data structure will do. However, trying to match a two element reduction to a scalar is undefined as is any match which is larger than the declared size of the coupled data structure. A multiple element data structure with ? number of elements is permissible though.

Couple statements are used to dynamically associate reductions to a specified data structure. Reductions are coupled to the structure specified in the last executed couple statement at the same or higher block level.

There is a system supplied routine which will produce a unique integer value for every unique character string supplied to it. The inverse function is also available. Normally, reductions will expect these unique integers to be stack entries rather than character strings as a considerable time and space saving can be realized. However, should a particular parsing job consist of short strings only, it might be profitable to bypass the string translation. Communicating the nature of stack contents is accomplished by the stack type, either NUMBER or STRING, associated with the stack specified by the appropriate couple statement. An example of a couple statement follows:

```
COUPLE(SYNXSTK);
```

Reductions match only a certain field within NUMBER stack elements extending from digit one to X where X will be defined by the implementation. Thus, the user may declare a wider or multiple dimensioned stack and gain parallel storage if he so desires. Stacks of type STRING will have the entire first element of each stack row matched by the reductions and consequently the only way to gain parallel storage here is with multiple dimensioning.

The left two sections of a reduction statement act as a pattern recognition and generation device for syntax parsing and bear little resemblance to the rest of CABAL even though reduction control is similar to IF-THEN statements. The third section, however, exhibits the full range of CABAL as its content is syntactically defined as a statement sequence, with the exception that another reduction statement may not appear unless imbedded within a BEGIN-END pair.

This structure, along with the placement freedom of reductions allows diverse ways to organize a language translator. Notably, the two ways most usually desired: a complete set of productions following one right after another with their associated semantic routines also grouped in one sequential mass disjoint from the reductions; and alternations of syntax and semantics statements with each syntactic mechanism containing or followed by its associated semantic routines.

VI. CODE GENERATION

Code generation is accomplished through the use of code statments and item statements. A code statement simply encloses in code brackets constructs which for the most part compose the CABAL language as a whole.

Item statements are used to provide parameters in conjunction with the indicated code from a code statement. Thus, the translator writer merely translates his input stream into valid CABAL and encloses it in code brackets.

There are a few restrictions and extensions which may appear in code brackets. With the exception of having to put out complete statements, structure is non-existent. Declarations and statements may be interspersed at will. A declaration in code brackets causes the generator to push the MST at the appropriate place and store the necessary information. Subsequent usage of declared items in statements causes retrieval of MST data in order to produce code.

The output from code brackets is a stream of items which are interpreted by the generator into code. Code is produced whenever a sufficient amount of information comes through the code brackets. Thus, one statement may be executed such as CODE{#1}, producing no code until the construct is sufficiently completed by, perhaps, CODE{← #1 + #2}.

All names within code brackets which must be identified have a syntax of #<integer>. This relates them to parameters in the most recently executed item statement. For example:

```
ITEM(SMT[4],SMT[1],#A,#B);
CODE{FOR#1← #4 TO #3 BY #2 DO};
```

Here we have the parameters in the item statement implicitly numbered starting with 1 at the left. The code statement designates which parameter it wants by signifying the number. The # in the item statement is used to indicate that a parameter represents a constant rather than a declared name.

When the generator receives a parameter representing a declared name, it looks in the MST to get the appropriate storage address and related information. When a parameter name has more than one declaration, the top one is always used.

Should it be necessary to pass a name parameter to the generator which has not yet been declared, like a label name for instance, the CHAIN function is useful. This function need only be used once per name as it continued to chain all instances of the name until the ASSIGN function is invoked. ASSIGN is used after the appropriate declaration has been made and retraces the chained list inserting appropriate addresses and completing any unfinished coding.

There are times when C^1_P wishes to transfer some of its data to C^0_P. This is facilitated quite easily as:

```
ITEM(RUNNAME,#TABLE);
CODE{#1← #2;};
```

This works if both data structures have the same declarations; TABLE is declared for the C^1_P level and during a compilation collects data which must be passed to the C^0_P where RUNNAME has been declared.

Control can be passed to compiled code by generating a GOTO <label name> NOW; where <label name> is some pre-declared label in the generated code. Return control to the point following the code statement which gave up control will happen if execution runs into a pre-generatied RETURN NOW. Execution of HALT by either C^0_P or C^1_P will give control to the operating system.

VII. SUMMARY

Looking now with an eye for comparing the language description with the design criteria, there are a number of points worth mentioning. For applicability to compiling-compilers there are included co-routines for natural phase and pass separation, reductions for syntax parsing, full algebraic power for semantics processing, structures and operators for string manipulation, system supplied routines for handling strings comfortably, a master symbol table easily accessible by both code generation system routines and user written semantic routines, and a code generation facility that couples the easy and nonchalant use of a high level language with optimal system routines capable of streamlining the resultant code to a degree determined by how much time is deemed worthwhile.

Producible translators include interpreters, which make use of the output facility for compile time data and code as well as reduction statements; conversational compilers, aided by ease of control flow between environment, compiler, and generated code; and multiple language systems using co-routine and reduction stack coupling. Compatability is maintained with a dynamic environment through machine independence and an open ended design ready to accept extensions. In this respect a macro facility is anticipated and seems reasonably easy to implement.

Modularity is provided not only by the language structure and its program and data structures, but also at a lower level its statement and declaration syntax is modular to a degree that makes extension trivial. Flexibility, besides benefiting from points already mentioned, is further assured by an absolute minimum of communication linkages and interdependencies within and among the language system subdivisions adn the system environment.

There is full algebraic power as well as the ability to reference all system variables and even drop down to an assembly language sequence. Readability and transparent learning is assured from the ALGOL nature as well as keeping the concepts to a minimum and using a high level language for code generation. The syntax was specially geared for terseness to such an extent that FOR-STEP-UNTIL-DO was superceded by FOR-TO-BY-DO.

Finally, reliability is facilitated through an easily understood and well-partitioned language system with no major hidden subdivisions.

In summation, I feel satisfied with the CABAL design for two reasons. First, I think it demonstrates that programming languages aimed at complex problems not fully formalized or understood need not wait for the absolute insight before user problems are considered. In fact, making the usage of such a system as painless as possible will do much for increasing the number of users and thereby quicken the time when language translation is yesterday's problem.

As a second point of satisfaction, I note that the results of the CABAL design phase are directly related to the goal system established. Hopefully, refined techniques for defining the initial goals of a design project will make this process quicken and the ultimate designing a matter of cause and effect.

References

1. Dove, R. K , Fierst, J., Shaw, M., McCreight, E., Adams, D., and Eve, J., CABAL Notes. Computer Center Reports #CCP-117, 176, 162, 191, 227, 229, 230, 232, 240, 250, Carnegie-Mellon University, Pittsburgh, Pa., 1966-68.

2. Feldman, J. A , A Formal Semantics for Computer Oriented Languages, Computation Center, Carnegie-Mellon University, Pittsburgh, Pa., 1964.

3. Krutar, R. A., Extensions to FSL, Proposed Extensions to FSL, and Methods and Techniques for Using FSL, Computation Center Report, Carnegie-Mellon University, Pittsburgh, Pa., July 20, 1965.

4. Iturriaga, R., Standish, T. A., Krutar, R. A. and Early, J. C., Techniques and Advantages of Using the Formal Compiler Writing System FSL to Implement a FORMULA ALGOL Compiler, Proc. AFIPS SJCC, 1966, pp. 241-252.

5. Cheatham, T. E. and Sattley, K., Syntax Directed Compiling, Proc. EJCC AFIPS, Vol. 25, 1964, pp. 31-57.

6. Dove, R. K., Co-routines - Control, Recursion, and Formalism, Proceedings of Modular Programming Symposium, Information and Systems Press, 14 Concord Lane, Cambridge, Mass. 02138, Sept. 1968.

7. Shaw, M. and Fierst, J., Design Criteria for Compiler-Compilers, CABAL Note 3, Computer Center Report #CCU-51, Carnegie-Mellon University, Pittsburgh, Pa., 1966.

8. Dove, R. K., A Modular Approach to Simulation and Language Design, Proceedings of Modular Programming Symposium, Information and Systems Press, 14 Concord Lane, Cambridge, Mass. 02138, Sept. 1968.

9. Naur, P., Program Translation Viewed as A General Data Processing Problem, Comm. A.C.M., Vol. 9, No. 3, March 1966.

10. Cheatham, T. E., Notes on Compiling Techniques, Computer Associates, Inc., Wakefield, Mass., 1965.

Charles M. Eastman
Department of Computer Science
Carnegie-Mellon University
July 1968

COMPUTER AUGMENTED DESIGN-A BIBLIOGRAPHY

The following bibliography covers those papers and publications that deal with the graphic, ill-defined or heuristic aspects of computer augmented design. It does not deal with traditional optimization or linear programming techniques.

The following outline is utilized. Where a particular study is relevant to several categories -- it has been listed more than once.

CLASSIFICATION OUTLINE

I. Design-general considerations
  - A. Definitions and descriptions of the design process
  - B. Design task analysis
  - C. Analysis of information retrieval systems for design

II. Operational computer programs augmenting design
  - A. Graphic output
  - B. Automated evaluation and simulation
  - C. Scheduling
  - D. Cost controls
  - E. Synthesis
  - F. Computer-aided design systems

III. Representations and models for design
  - A. Morphological and syntactic analysis of natural design languages
  - B. Representations of surfaces, spaces, and forms
  - C. Possible representations
    - 1. Graphs
    - 2. Lists
    - 3. Associative nets
    - 4. Numerical systems

IV. Design operations
  - A. Planning
  - B. Heuristic search
    - 1. Theory
    - 2. Examples
  - C. Processes that learn
  - D. Pattern recognition
  - E. Other operational forms

V. Applications
  - A. Computer design
  - B. Space allocation
  - C. Structures
  - D. Urban models
  - E. Transportation
  - F. Circuits and Logic
  - G. Other

Bibliography

1. Ackoff, Russell L., Scientific Method: Optimizing Applied Research Decisions, Wiley, New York, 1962. (Chapters 3, 4, 5 IA)

2. Aicher, J. R., "Producing piping isometric drawings via computer plotter", in Computer Bulletin, 11, 2; 134-138 (Sept. 1967).

3. Alexander, C and Manheim, M. L., "The Use of Diagrams in Highway Route Location: An Experiment", publication no. 161, Report R62-3, Civil Engineering Systems Laboratory, M.I.T., 1962. (IIE, VE)

4. Alexander, C., "HIDECS3: Four Computer Programs for the Hierarchical Decomposition of Systems Which Have an Associated Linear Graph", M.I.T. Department of Civil Engineering Report No. R63-27, June 1963. (IIIC1, IVE)

5. Alexander, Christopher, "The Theory and Invention of Form", in Architectural Record, April 1965, pp. 177-186. (IIIC1, VB, VD)

6. Alexander, Christopher, "The Pattern of Streets", American Institute of Planners Journal, Sept. 1966, pp. 273-278. (IIIC4, VE)

7. Alexander, Christopher, Notes on the Synthesis of Form, Harvard University Press. (IA, IIE, IIIC1, IVA, VB, VD)

8. Alger, J. R., Hays, C. V., Creative Synthesis in Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1964. (Chapter 2 and Bibliography, IA)

9. Archer, L. Bruce, Systematic Method for Designers, reprinted from Design magazine, London, 1965. (IA)

10. Armour, G. C. and Buffa, E. S., "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities", Management Science, Jan. 1963, pp. 294-309. (IIB, IIIB, IVB1, VB)

11. Asimow, M., Introduction to Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1962. (Chapters 1-9 IA)

12. Au, Tung, "Heuristic Games for Structural Design", J. Struct. Div. Proc. ASCE ST6, Dec. 1966. (IIB, VC)

13. Au, T., Recker, W. W., "Engineering Synthesis Game: Simple Structural Framing in a Lunar Environment", Report No. ES-1, Civil Engineering Department, Carnegie Institute of Technology, 196 . (IIB, VC)

14. Au, T., Parti, E. W., "Building Construction Game: General Description", Report No. BC-1, Civil Engineering Department, Carnegie Institute of Technology, 1966.

15. Bellman, Z., Kalaba, R., Zadah, L., "Abstraction and Pattern Classification", J. Math. Analy. and Applic., 13,1, Jan. 1966, p. 1-7. (IIIC4, IVD)

16. Bellman, R. and Brock, "On the concepts of a problem and problem solving", Amer. Math. Month., Feb., 67, 2, 119-34.

17. Berge, C., Theory of Graphs and their Applications, translated by Allison Doig, Wiley, New York, 1962.

18. Bernholtz, A., and Bierstone, E., "Computer-Augmented Design", in Design and Planning II, M. Krampen, ed., Hastings House, New York, 1967. (IIIC1)

19. Beshers, James M. (ed.), Proc. Conf. Computer Methods in the Analysis of Large-Scale Social Systems, Cambridge, Mass., Oct. 1964, Joint Center for Urban Studies, M.I.T. and Harvard University, Cambridge, Mass., 1965, 207 pp.

20. Black, W. L., "Discrete Sequential Search", in Info. and Control, 8,2, April 1965, pp. 159-162. (IIIC4, IVE)

21. Bobrow, Daniel, "Problems in Natural Language Communication with Computers", IEEE Trans. on Human Factors, Vol. HFE-8, No. 1, March, 1967.

22. Bobrow, D. G., Raphael, B., "A Comparison of List-Processing Languages", Comm. ACM, April, 1954.

23. Bolt, Beranek and Newman, Inc., Computer-Aided Checking of Design Documents for Compliance with Regulatory Codes, distributed by Clearinghouse, National Bureau of Standards, Springfield, Virginia, 22151, Document No. PB174 095. (VB)

24. Bolt, Beranek and Newman, Inc., Development of Notation and Data Structure for Computer Applications to Building Problems, distributed by Clearinghouse, National Bureau of Standards, Springfield, Va. 22151, Document No. PB 174, 795, March 1967. (IIIA, VB)

25. Bowen, Hugh M., "Rational Design" serial in Indust. Design, Feb., March, April, May, June, July, Aug., 1964. (IA)

26. Brown, S. A., Drayton, C. E., Mittman, B., "A Description of the APT Language", Communications ACM 6, 11, Nov. 1963. (IIIB)

27. Buck, C. Hearn, Problems of Product Design and Development, Pergamon Press, London, 1963. (Not worth reading)

28. Buffa, E. S., Armour, G. C., Vollman, T. E., "Allocating Facilities with 'Craft' ", in Readings in Production and Operations Management, E. S. Buffa ed., Wiley, New York, 1966. (IIB, IIIB, IVB1, VB)

29. Busacker, R. G. and Saaty, T. L., Finite Graphs and Networks, McGraw-Hill, New York, 1965.

30. Chomsky, Naom, The Aspects of the Theory of Syntax, M.I.T. Press, Cambridge, 1965. (IIIA)

31. Clark, Welden E., Souder, James J., Planning Buildings by Computer, A and E News, pp. 25-33, March, 1965.

32. "Computer-aided Design", Product Engineering Magazine, Mono #R113, McGraw-Hill, New York, 1960-1965.

33. Coons, S. A., "Computer Graphics in Innovative Engineering Design", in Datamation, May 1966, pp. 32-34.

34. Coons, S. A., "An Outline of the Requirements for a Computer-aided Design System", Proceedings SJCC, 1963, Spartan Books, Washington, D.C. (IB, IIF)

35. Coons, S. A., "Computer-aided Design", in Design and Planning II, M. Krampen, ed., Hastings House, New York, 1967. (IA, IIA, IIIB)

36. Coons, S. A., "Surfaces for Computer-aided Design of Space Forms", Project MAC, Report MAC-TR-41, DDC Report AD 663 504, June 1967. (IIIB)

37. Cooper, L., "Heuristic Methods for Location-Allocation Problems", SIAM Review, 6, 1, Jan. 1964, pp. 37-53. (IIE, IIIC4, IVB1, VB)

38. Cooper, W. S., "Fact Retrieval and Deductive Question Answering Information Retrieval Systems", Journal ACM, 11, 2, pp. 117-137, 1964. (IC)

39. Davis, R. M., "Classification and Evaluation of Info. System Design Techniques", in Proc. 2nd Cong. Info. Systems Science, pp. 77-83.

40. Dent, Colin, Quantity Surveying by Computer, Oxford University Press, 1964, paperback.

41. Dougald, D. E., The Development of a Data Storage and Retrieval System For Building Envelope Design Information, final report of research study, Project MODCON - Subtask Beta., Report 67-3, Pennsylvania State University Inst. for Building Research, July 1967.

42. Eastman, Charles, "Explorations of the Cognitive Processes in Design", Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., Feb. 1968, ARPA Document DDC No. 671158. (IA, IB, IC, IIIA, IVB2, VB)

43. Eastman, C., "On the Analysis of Intuitive Design Processes", Proceedings of the First International DMG Conference, M.I.T., June, 1968. (IA, IIE, IIIA, IVB2, VB)

44. Eder, W. E., Gosling, W., Mechanical System Design, Pergamon Press, London, 1965. (CHAPTERS 1, 2, 3, 4 IA)

45. Educational Facilities Laboratories, "School Scheduling by Computer: The Story of GASP", Educational Facilities Laboratories, New York, 1964. (IIC)

46. Ernst, George, Newell, Allen, "Some Issues of Representation in a General Problem Solver", Proceedings SJCC, 1967, Spartan Books, Washington, D. C. (IIE, IIIA, IIIC4, IV1)

47. Evans, D. H., "Modular Design - A Special Case of Non-Linear Programming", in Op. Res., 11, 4, pp. 637-647, July-Aug., 1963.

48. Fair, G. R. et al, "Note on the Computer as an Aid to the Architect", in Computer J., June 1966. (IIB, VB)

49. Falk, H., "Computer Programs for Circuit Design", Electro-Techn, 77, 6, pp. 54-7, June 1966. (IIB, IIE, VF7

50. Feder, J., Linguistic Specification and Analysis of Classes of Patterns, Technical Report 400-147, U. S. Department of Commerce (1966). (IIIA)

51. Fenves, S. J., Logcher, R. D., Mauch, S. P., STRESS: A Reference Manual, Cambridge, Mass., M.I.T. Press, 1965. (IIF)

52. Fetter, William A., Computer Graphics in Communication, McGraw-Hill, New York, 1965. (IIA)

53. Flagle, C. D., Higgens, W. H., Roy, R. H., Operations Research and Systems Engineering, J. Hopkins Press, Baltimore, 1960.

54. Fletcher, J. C., "A Program to Solve the Pentomino Problem by the Recursive Use of Macros", Comm. ACM, 8, 10, Oct. 1965, pp. 621-625.

55. Fogel, Lawrence J., Biotechnology: Concepts and Applications, Prentice-Hall, Englewood Cliffs, New Jersy, 1963. (Chapters 1, 2 - IIIC4)

56. Forrester, J. W., Industrial Dynamics, Wiley, New York, 1961.

57. Freimer, M. and Simon, L. S., "The Evaluation of Potential New Product Alternatives", in Manag. Science, 13, 6, Feb. 1967, pp.B-279. (IIB, V9)

58. French, Carroll D., An Electronic Forms File: New Assistance for Legal Drafting, ABAJ 50, 1 Jan. 1964, 41-43.

59. Freund, Louis and Sodosky, "Linear Programming Applied to the Optimization of Instrument Panel and Workplace Layout", Human Factors, 9, 4, Aug. 1967, pp. 295-300.

60. Frisch, R., Inseeking by the Nonplex Method in Non-Linear Programming, Inst. Natl. Planning, Documentation Center, Cairo, Egypt, Memo. 396, Feb. 1964, 14 pp.

61. Gagné, R. M., Ed., Psychological Principles of System Development, Holt, Rinehart and Winston, New York, 1962. (Chapters 5, 6 IB)

62. Gall, D. and Krokosky, "A Generalized Procedure for Automated Optimal Design", Proc. 4th National Conf. on Engr. Design, Dartmouth, 1967.

63. Garfield, E., "Diagonal Display: A New Technique for Graphic Representation of Complex Topological Networks", Institute for Scientific Information, DDC No. AD 664 059, Sept. 1967. (IIIB, IIIC1)

64. Garin, Robert A., "A Matrix Formulation of the Lowry Model of Intermetropolitan Activity Allocation", J.A.I.P., Vol. 31, 6, Nov. 1966, pp. 361-366.

65. Ginzburg, Seymour, The Mathematical Theory of Context-free Languages, McGraw-Hill, 1966.

66. Goode, Harry H. and Machol, Robert., System Engineering: Introduction to the Design of Large Scale Systems, McGraw-Hill, New York, 1957. (Chapters Part I IA, Part III, Chapters 21, 27 IB)

67. Gordon, J. J., Synectics, Collier Books, New York, 1968. (IC)

68. Gore, William I., Administrative Decision Making: A Heuristic Model, Wiley, New York, 1964.

69. Gosling, W., The Design of Engineering Systems, Haywood, London, 1962. (Chapters 1, 3 IA; Chapter 2 IIIC1; Chapter 5 IVB2)

70. Grason, John, "A Dual Graph Representation for Space Filling Location Problems of the Floor Plan Type", Carnegie-Mellon University, paper presented at the DMG Conference, M.I.T., June, 1968. (IIE, IIIC1, VB)

71. Gray, J. C., "Compound Data Structure for Computer-aided Design: A Survey", Proc. ACM National Meeting, Spartan Press, Washington, D.C., 1967. (IIF, IIIC1)

72. Green, B. F., Jr., Wolf, A. K., Chomsky, C., Loughery,"BASEBALL: An Automatic Question-Answerer,"Proceedings Western Joint Computer Conference, May 1961, pp. 219-224.

73. Green, Claude C., "Research on Intelligent Question-Answering System," Stanford Research Institute Report, AD 656789, May 1967.

74. Greenberger, C. B., "The Automatic Design of a Data Processing System", in IFIP Congress 65 Vol. 1, 277-82.

75. Gregory, S., The Design Method, London, Butterworth, 1966.

76. Grijda, Nico H., "Problems of Computer Simulation", Behav. Sc., Jan. 1967.(IB)

77. Gross, Leonard D., "Coding Clinical Laboratory Data for Automatic Storage and Retrieval, Comm. ACM 6, 11, Nov. 1963, 690-694.

78. Guetzkow, Harold (ed.), Simulation in the Social Sciences: Readings, Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

79. Haney, Frederick, M., "Using a Computer to Design Computer Instruction Sets", doctoral thesis Carnegie-Mellon University, Pittsburgh, Pa., June, 1968. (IA, IIE, VA)

80. Harris, Britton, "Plan or Projection: An Examination of the Use of Models in Planning", A.I.P.J., 26, 4, Nov. 1960, pp. 265-272.

81. Harris, Britton, "Urban Development Models" A.I.P.J., Vol. 31, No. 2.

82. Herman, Robert and Gardels, Keith, "Vehicular Traffic Flow", in Scientific American, 209, 6, Dec. 1963, pp. 35-43.

83. Hershdorfer, A., Understanding the Architectural Design Process Through An Analysis of its Information Structure, presented at the Conference on Computers in Architecture sponsored by Computer Education Center, Pratt University, New York Chapter AIA, New York, May 23, 1966.

84. Hill, Lawrence, S., "Some Implications of Automated Design on the Engineering Process", Rand Corporation, DDC No. AD655 965, July 1967. (IB)

85. Holstein, David, and Smith, Christopher F., "Automating Design Procedures with Decision-Making Systems", Data System Design, 1, 3, March 1964, 15-20.

86. Holstein, David, "Automated Design Engineering,"Datamation, 10, 6, June 1964, 28-34.

87. Hufschmidt, Maynard M., Simulating the Behavior of a Multi-Unit, Multi-Purpose Water-Resource System, symposium on simulation models; methodology and applications to the behavioral sciences, 203-220, South-Western Publishing Co., Cincinnati, Ohio, 1963, pp. 289.

88. Hutchinson, B. G., "Simulation of Exhibition Visitor Circulation on a Digital Computer", in Design and Planning II, M. Krampen ed., Hastings House, New York, 1967. (IIB, VB)

89. Jacks, et al, "The GM DAC-I System: Design Augmented by Computers", General Motors Research Laboratories Report GMR-436, Oct. 1964. (IIF, IIIB)

90. Jay, L. S., "A Systematic Approach to the Problems of Town and Regional Planning", in Conference on Design Methods, J. C. Jones and D. G. Thornley eds., Macmillan Co., New York, 1963. (IA, VD)

91. Jensen, Paul A., "A Graph Decomposition Technique for Structuring Data", Report No. 106-3, Computer Command and Control Co., DDC No. AD 658 756.

92. Jerger, J. J., Systems Preliminary Design: Principles of Guided Missile Design, D. van Nostrand, New Jersey, 1960, Chapter 10. (IB, IIIC4, VG)

93. Johnson, Lee H., Engineering: Principles and Problems, McGraw-Hill Book Co., New York, 1960, general problems emphasizing analysis. (No design theory)

94. Johnson, Timothy E., "Sketchpad III: A Computer Program for Drawing in Three Dimensions", in Proceedings Spring Joint Computer Conf., 1963, Spartan Press, pp. 347-353. (IIA, IIIB)

95. Jones, J. C., "A Method of Systematic Design", in Conference on Design Methods, J. C. Jones and D. G. Thornley eds., Macmillan Co., New York, (IA, IIE, IIIC)

96. Jordan, Jane C., editor, ICES: Programmers' Reference Manual, Report R67-50, October 1967, M.I.T.

97. Karnapp, D. C., "Random Search Techniques for Optimization Problems", Automatica, Vol. 1, No. 2-3, 1963.

98. Kirsh, R. A., "Computer Interpretation of English Text and Picture Patterns", IEEE Trans. Elect. Comp., Aug. 1964. (IIIA, IIIB)

99. Kish, Leslie, Survey Sampling, John Wiley and Sons, New York, 1965, pp. 643.

100. Kleinmuntz, B., Problem Solving: Research Method and Theory, Wiley, 1966.

101. Klerer, M. and May, J., "A User Oriented Programming Language", Computer J., July 1965, pp. 103-108. (IIF)

102. Knowlton, K. C., "Computer Generated Movies, Designs and Diagrams", in Design and Planning II, Hastings House, New York, 1967. (IIIB)

103. Knuth, Donald A., The Art of Computer Programming, Vol. I, Addison-Wesley, Reading, Mass., 1968.

104. Krampen, Martin, "Design as Creative Problem Solving", in Design and Planning, M. Krampen ed., Hastings House, New York, 1965. (IB)

105. Krauss, Richard, Myer, John, "Design: A Case History", Center for Building Research, M.I.T., 1968. (IB, VB)

106. Kreidberg, M. B., Field, Highlands, et al, "Problems of Pediatric Design", Research Project NM00235, U. S. Public Health Service, Tufts Medical Center, November, 1965.

107. Krick, Edward V., An Introduction to Engineering and Engineering Design, John Wiley, New York, 1965. (IA)

108. Krokosky, Edward M., "The Ideal Multifunctional Constructional Material", J. Struct. Division Proc. Amer. Soc. Civ. Engrs., ST4, April 1968, Vol. 94. (IIE, IIIC4, VC)

109. Lang, C. A., Polansky, R. B., Ross, O. T., "Some Experiments with an Algorithmic Graphical Language", Tech. Memo ESL-TM-200, Elect. Systems Lab., M.I.T., Cambridge, August, 1965.

110. Lange, Oskar, Wholes and Parts: A General Theory of System Behavior, Pergamon Press, New York, 1965, E. Lepa (trans.).

111. Ledley, R. A., "Concept Analysis by Syntax Processing", Proc. of Amer. Doc. Inst. Annual Meeting, Vol. I, 1964.

112. Ledley, R. S , Programming and Utilizing Digital Computers, McGraw-Hill, 1963, Chapters 6-8.

113. Lee, Vernon, Ball, H. G., Wadsworth, et al, "Computerized Aircraft Synthesis", in J. Aircraft, 4, 5, 402-408, Sept.-Oct. 1967.

114. Leondes, C. T. (ed.), Advances in Control Systems: Theory and Applications, Vol. 1, Academic Press, New York, pp. 365, 1964.

115. Liming, R. A. and Harris, H. R., "Autodraft - A System for Computer Aids to Design and Drafting", Proc. 4th Annula Meeting of UAIDE.

116. Logcher, Robert D., Biggs, John M., ICES: STRUDL1, Structural Design Language, General Description, Report R67-55, Sept. 1967, M.I.T.

117. Logcher, Robert D. et al, ICES STRUDL1, The Structural Design Language, User's Manual, Report R67-56, Sept. 1967, R. A. M.I.T.

118. Lorens, Charles S., Flowgraphs - for the Modeling and Analysis of Linear Systems, McGraw-Hill, New York, pp. 178, 1964, paperback.

119. Lowenbach, F. C., "Information Science - Man and Machine", Synthesis 19, 217-218 (June-July 1964), 218-234.

120. Lowry, Ira S., "A Short Course in Model Design", A.I.P.J., pp. 158-166, May 1965. (IB, IIB, VD)

121. Lynch, Kevin, "Quality in City Design", in Who Designs America?, L. B. Holland, ed., Doubleday Anchor, pp. 120-171, 1965. (IA)

122. Malmberg, B., Structural Linguistics and Human Communication, Academic Press, New York, 1963.

123. Manheim, Marvin, Hierarchical Structure: A Model of Design and Planning Processes, M.I.T. Report No. 7, M.I.T. Press, 1966. (IA, IB, IIB, IVB, VE)

124. Manheim, M., "Problem Solving Processes in Planning and Design", M.I.T. Department of Civil Engineering Report No. P67-3, Jan. 1967. (IA)

125. Manheim, Marvin, "Principles of Transport System Analysis", M.I.T. Department of Civil Engineering Report P67-1, Jan. 1967. (IB, VE)

126. Marienfeld, Horst, Simulation, Fundamentals and Application to Design and Development of Structural Aircraft - Bibliography 1934-63, VDI, Dusseldorf, Germany, pp. 888, 1964 (German).

127. Maron, M E., On Cybernetics, Information Processing, and Thinking, RAND Corporation, Santa Monica, California, P -2879, p. 41, March 1964.

128. Martin, Francis F., Computer Modeling and Simulation, Wiley, New York, 1968. (IIB)

129. Martino, R. L., Finding the Critical Path, American Management Association, p. 144, 1964.

130. Mashkovich, S. A., "The Use of High-Speed Digital Computers for Planning the Development of the Network of Aerological Stations", Metereol. igldrologiya 7 (1963), 3-9 (Russian), Ref. Zh. Mat. 9 (Sept 1963), Rev. 9V362.

131. McCarn, D. B., "Large Scale System Design Techniques", in Proc. 2nd Congress Inf. and System Sciences, pp. 95-98.

132. Meissel, Peter, Probability Model of a Signal Controlled Multi-Lane Intersection, pts. 1, 2, Math. Tech. Wirts. 10, 1 (1963), 1-4; 10, 2 (1963) 63-68 German.

133. Meister, David and Farr, Donald, "The Utilization of Human Factors Information by Designers", Human Factors, 9, 1, p. 71-89, Feb. 1967. (IB, IC)

134. Miles, L. D., Technique of Value Analysis and Engineering, McGraw-Hill, 1962.

135. Miller, C. M., Man Machine Communication in Civil Engineering, M.I.T. Civil Engineering Department Report T63-3, June 1963.

136. Miller, G. A., Galenter, E., Pibram, Plansand the Structure of Behavior, Holt, New York, 1960. (IB, IC)

137. Miller, R. B., "Task Description and Analysis", in Psychological Principles of System Development, R. M. Gagni et al (eds.), Holt, New York, 1962. (IB)

138. Milne, Murray, "Architectural Applications of Computer Based Network Analysis Models", A.I.A. Researchers Conference, Galtinburg, Tenn., October 1967. (IIIC1; VB)

139. Minsky, M., "Heuristic Aspects of the Artificial Intelligence Problem", AD 236885, Lincoln Labs. M.I.T., 1965. (IV B1)

140. Minsky, Marvin, "Steps Towards Artificial Intelligence", in Computers and Thought, Feigenbaum and Feldman (eds.) McGraw-Hill, New York, 1963. (IIIA, IVB1)

141. Moran, Thomas P., "A Model of a Multi-Lingual Designer", Carnegie-Mellon University paper presented to the DMG Conference, M.I.T., June 1968. (IIIB, IIIC, VB)

142. Morgan, H. L., "The Generation of a Unique Machine Descr. for Chemical Structures - A Technique Developed at Chemical Abstracts Service," J. Chem., Doc. 5, 2 pp. 107-113, May 1965.

143. Morse, R. W., Arnberg, J. E., Jonas, J. L., "Seattle Viaduct Redesigned Using C.P.M.,"Civil Engineering 34, 10, pp. 46-47, Oct. 1964.

144. Myer, T. H., "Computer-aided Cost Estimating Techniques for Architects", Cambridge, Mass., U. S. Department of Commerce Clearinghouse No. PB174098, 1966. (IIB, IID, VB)

145. Narasimhan, R., "Syntactic Description of Pictures and Gestalt Phenomena of Visual Perception", Report No. 142, Digital Computer Laboratory, University of Illinois, Urbana, Ill., July 1963. (IIIA, IIIB)

146. Narasimhan, R., "A Linguistic Approach to Pattern Recognition", Report No. 121, Digital Computer Laboratory, University of Illinois, Urbana, Ill., July 1962. (IIIA, IIIB)

147. Newell, A., Shaw, J. C., Simon, H. A., "A Variety of Intelligent Learning in a General Problem Solver", in Yovits, M.C. and Cameron, S. (eds.), Self-Organizing Systems.

148. Newell, A., Ernst, G., "The Search for Generality", Proc. IFIP Congress 1965, Spartan Books, Washington, D. C., 1965.

149. Newell, Allen, "Studies in Problem Solving: Subject 3 on the Cryptarithmetic Task", Department of Computer Science Report, Carnegie Institute of Technology, Pittsburgh, Pa., July 1967. (IB, IVB1, IVB2)

150. Newell, A. and Simon, H. A., "Computer Simulation of Human Thinking", in Science 134, 3495, 22 Dec. 1961, pp. 2011-2017. (IB, IVB)

151. Newell, Allen, "Limitations of the Current Stock of Ideas about Problem-Solving", Conf. on Electronic Information Handling, A. Kent, O. Taulbee (eds.) Spartan Press, Washington, D. C., 1965. (IIIA, IVB1)

152. Newell, Allen, Simon, H. A., "GPS: A Program that Simulates Human Thought", in Computers and Thought, Feigenbaum and Feldman (eds.) McGraw-Hill, New York, 1963. (IIE, IVB1)

153. Newell, Allen, "The Possibility of Planning Languages in Man-Computer Communication", in Communication Processes, Pergamon Press, New York, 1964. (IB, IC, IVA)

154. Newman, W. M., "An Experimental Program for Architectural Design", Computer J., June 1966. (IIA, VB)

155. Noll, A. M., "Computers and the Visual Arts", in Design and Planning II, M. Krampen, ed., Hastings House, New York, 1967. (IIIB)

156. Norris, K. W., "The Morphological Approach to Engineering Design", in Conference on Design Methods, J. C. Jones and D. G. Thornley, eds., Macmillan Co., New York, 1963. (IA, IIIA, IVE)

157. O'Brien, James J., CPM in Construction Management, McGraw-Hill, New York, 1965. (IIC)

158. Overton, R. K., "Intelligent Machine and Hazy Questions", Comp. and Auton. 14, pp. 26-30, July 1965.

159. Parnas, David L. and Darringer, John, "SODAS and a methodology for system design", Proceedings Fall Joint Computer Conference 1967, Spartan Press, Washington, D. C. (IVA, VA)

160. Parnas, D. L., "A Language for Describing the Functions of Synchronous Systems", in Communic. ACM 9,2, pp. 71-77, Feb. 1966. (IIIA)

161. Pottle, C., "State-Space Techniques for General Active Network Analysis", in System Analysis by Digital Computer, F. F. Kuo and J Kaiser (eds.), Wiley, New York, 1966.

162. Quillian, M. Ross, "Semantic Memory", Bolt, Berenek and Newman, AD641 671, Oct. 1966. (IC: IIIC3)

163. Raphael, B., "SIR: A Computer Program for Semantic Information Retrieval", doctoral thesis, M.I.T., Cambridge, Mass., June 1964. (IC)

164. Raphael, Bertram, "A Computer Program which 'Understands'," in Proc. AFIPS 1964 Fall Joint Comput. Conf., pp. 577-589; see main entry CR Rev. 7102. (IC)

165. Redding, R. J., "Automation in the Detailed Design of Chemical Plant", Control, 11, 108, pp. 275-279, June 1967.

166. Reitman, Walter, "Heuristic Decision Procedures, Open Constraints, and the Structure of Ill-Defined Problems", in Human Judgments and Optimality, M. W. Shelly and G. L. Bryan (eds.), Wiley, New York, 1964. (IA, IIE, IVB1)

167. Roe, P. H., Soulis, G. N., Handa, The Discipline of Design, Adlyn and Bacon, Boston, 1967.

168. Rogers, Andrei, "Theory of Intraurban Spatial Structure: A Dissenting View", in Land Econ., Feb. 1967.

169. Roos, Daniel, ICES System: General Description, M.I.T. Press, 1967. (IIF)

170. Roos, Daniel, ICES Systems Design, M.I.T. Press, 1966. (IIF)

171. Roos, Daniel; Miller, C. L., COGO-90: Engineering User's Manual, M.I.T., Cambridge, Mass., Res. Report R64-12, p. 69, April 1964. (IIF)

172. Ross, D. T. and Rodriguez, J. E., "Theoretical Foundations for the Computer-aided Design System", Proceedings SJCC, 1963, Spartan Press, Washington, D. C. (IC, IIF, IIIC1)

173. Ross, Douglas T., "The AED Approach to Generalized Computer-Aided Design," Proc. 22nd National Conference, Association for Computing Machinery, P-67, Washington, D. C., hompson Book Co., pp. 367-385, 1967. (IIF, IIC1)

174. Schmit, L. A., "Structural Design by Systematic Synthesis", 2nd Conf. on Elect. Comp., Pittsburgh, Pa., 1960. (IIIC4, VC)

175. Schmit, L. A., Morrow, W. M., "Structural Synthesis with Buckling Constraints", J. Struct. Div. ASCE ST2, April 1963. (IIIC4, VC)

176. Schmit, L. A., Kicher, F. P., "Synthesis of Material and Configuration Selection", J. Struct. Div. ASCE, ST3, June 1962. (IIIC4, VC)

177. Schwieg, Zeev, "An Application of the Dynamic Programming Method to the Planning of the Faleghan Conduit in Iran", in Proc. 3rd National (Israeli) Conference on Data Processing, Israel, 1967.

178. Sebestyen, George S., Decision-Making Process In Pattern Recognition, Macmillan Co., New York, 1962, CR 4, 4 (July-Aug. 1963) Rev. 4301, ACM monograph. (IIIC4, VD)

179. Shapiro, G. and Rogers, M. (ed.), Prospects for Simulation and Simulators of Dynamic Systems, Spartan Books, New York, 1967.

180. Shelly, M. W. and Bryan, G. L., Human Judgments and Optimality, Wiley, New York, 1964. (Chapter 1, IIIA)

181. Sheng, C. L., "Threshold Logic Elements Used as a Probability Transformer", Journal of the Association for Computing Machinery, Vol. 12, No. 2, pp. 262-276, 1965.

182. Sides, C. David, Jr., "A Computer Based Cost Analysis for the Building Designer - IBIS II," presented at the SICCAP Technical Session, Fall Joint Computer Science Conference, Anaheim, Calif., Nov. 1967. To be published in ACM-SICCAP Bulletin, Association for Computing Machinery, New York.

183. Silvestri, L. G., "Computer Correlation of Micro-organisms", in Enslein, Kurb. (ed.) Data Acquisition and Processing in Biology and Medicine, Vol. 2, pp. 43-53.

184. Simmons, R. F., Londe, D., "NAMER: A Pattern Recognition System for Generating Sentences about Relations Between Line Drawing", Doc. TM-1798, System Development Corp., Santa Monica, Calif., 1964.

185. Simon, H. A., "Representation in Tic-Tac-Toe", C.I.P. Paper #90, Carnegie Institute of Technology, (ditto copy) June 1966. (IIIA)

186. Simon, H. A., "The Sciences of the Artificial", unpublished manuscript, 1968. (IA, IVB)

187. Simon, Herbert A., Newell, Allen, "Information Processing in Computer and Man", American Scientist 52, 3, pp. 281-300, Sept. 1964.

188. Sinden, Frank W., "Principles and Programming of Animation", in Design and Planning II, M. Krampen ed., Hastings House, New York, 1967. (IIIB)

189. Slagle, J. R., "An Efficient Algorithm for Finding Certain Minimum Cost Procedures for Making Binary Decisions", J. ACM, pp. 253-264, 1963. (IVB1)

190. So, H. C., "Analysis and Iterative Design of Networks Using On-Line Simulation", in System Analysis by Digital Computer, F. F. Kuo and J. Kaiser (eds.), Wiley, New York, 1966.

191. Soss, Margo A., Wilkinson, William D. (eds.), Computer Augmentation of Human Reasoning, Washington, D. C., 1964, Spartan Books, Washington, D. C., 1965, pp. 235.

192. Souder, J. J., Clark, W. E., Planning for Hospitals: A System Approach Using Computer-Aided Techniques, American Hospital Assoc., 1964. (Chapter 2, IA; Appendix A, IIB; All, VB)

193. Souder, James J., Estimating Space Needs and Costs in General Hospital Construction, Amer. Hospital Assoc., Chicago, 1963.

194. Soulis, G. N. and Ellis, J., "The Potential of Computers in Design Practice", in Design and Planning II, M. Krampen ed., Hastings House, New York, 1967. (IB)

195. Standish, T., A Data Definition Facility for Programming Languages, Ph.D. thesis, Carnegie Institute of Technology, 1967.

196. Stark, Lawrence, Dickson, James F., "Remote Computerized Medical Diagnostic Systems", Comput. Autom. 14, 7, pp. 19-21, July 1965.

197. Starr, Martin, Product Design and Decision Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1963. (IIB)

198. Starr, Martin K., "Planning Models", Manage. Science, 13, 4, Dec. 1966. (IA, IVA)

199. Steger, Wilbur, "The Pittsburgh Urban Renewal Simulation Model", J. Amer. Inst. Planners, pp. 144-150, May 1965. (IIB, VD)

200. Steger, Wilbur A., "Review of Analytic Techniques for CRP", A.I.P.J., pp. 166-172, May 1965. (IIB, VD)

201. Stephenson, F. J., "Performance Concepts for Building Technology", Kansas University, U. S. Department of Commerce Clearinghouse, No. PB174 095, Sept. 1965.

202. Stone, Richard, A Computable Model of Economic Growth, M.I.T. Press, Cambridge, Mass., pp. 91, 1964.

203. Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System", in Proc. Spring Joint Comp. Conf., Spartan Press, Washington, D. C., pp. 329-346, 1963. (IIA)

204. Sutherland, I. E., "Computer Graphics: Ten Unsolved Problems", Datamation, pp. 22-27, May 1966. (IIA)

205. Sutherland, R. L., Engineering Systems Analysis, Addison-Wesley Pub. Co., Reading, Mass., 1958.

206. Sutherland, W. R., "The On-line Graphical Specification of Computer Procedures", M.I.T. Lincoln Laboratory Tech. Report No. 405.

207. Tanimoto, T, and Loomis, R. G., "The Application of Computers to Clinical Medical Data", Proc. 1st IBM Medical Symp., Poughkeepsie, New York, June, 1959, pp. 93-184.

208. Teague, L. C., Hershdorfer, A., "BUILD - An Integrated System for Building Design", Proc. ASCE Struct. Engr. Conf., Seattle, Wash., May, 1967. (IIF, IIIB, VB)

209. Thomsen, Charles, "How One Office Uses Computers", Architecture and Engineering News, June 1966. (IIIC4, VB)

210. Tomovic, Rajko, Sensitivity Analysis of Dynamic Systems, McGraw-Hill Book Co., pp. 142, New York, 1964.

211. Tonge, F. M., A Heuristic Program for Assembly Line Balancing, Prentice-Hall, Englewood Cliffs, New Jersey, 1961.

212. Vigor, D. B., "Data Representation - the Key to Conceptualization", Mach. Intell., 2, pp. 33-44, 1968.

213. Walsh, Brian F., "The Role of Specification in Design", Fourth National Conference on Engineering Design, Dartmouth, 1967. (IA)

214. Watt, Kenneth, E. F., "Computers and the Evaluation of Resource Management Strategies", Amer. Scient. 52, 4 pp. 408-418, Dec. 1964.

215. Watt, W. C., "Structural Properties of the Nevada Cattlebrands", in Computer Science Research Review, 1967, Carnegie-Mellon University, Pittsburgh, Pa. (IIIA)

216. Webber, M. M., "The Roles of Intelligence Systems in Urban-Systems Planning", J.A.I.P., Nov. 1968, Vol. 31, 4, pp. 289-96.

217. Wei, M. L. C., Au, T., "Bridge Design Game", Report No. BD-3, Civil Engineering Department, Carnegie Institute of Technology, 1967.

218. Wells, R. A., Jr., "ICES STRUDL-1, The Structural Design Language, the Uses of ICES STRUDL-1, Report R67-57, M.I.T., Sept. 1967.

219. Wilde, D. J., Optimum Seeking Methods, Prentice-Hall, 1964.

220. Wilde, D. J., Beightler, C. D., Foundations of Optimization, Prentice-Hall, 1967.

221. Willis, David G., "Strategies of Function Decomposition for Artificial Intelligence", Vol. 2, DDC Report No. AD620-186, Computer Usage Co., July 1965. (IIIC1)

222. Wilson, W. E , Concepts of Engineering System Design, McGraw-Hill, New York, 1965.

223. Wong, A. K. C., "Toward Artificial Intelligence in Fluid Mechanics", Ph.D. thesis, Carnegie-Mellon University, 1968.

224. Wong, A. K. C., Bugliarello, G., "An Artificial Intelligence Approach for Fluid Mechanics" (Abstract), Proceedings of 1967 Canadian Congress.

225. Wood, C. F., "Review of Optimization Techniques", IEEE Trans. of Systems Sc. and Cybernetics, V. SSC-1, pp. 14-20, Nov. 1965.

226. Woodson, Thomas T., Introduction to Engineering Design, McGraw-Hill, New York, 1966. (Chapter 2, IB; Chapter 3, IC)

227. Zadah, L. A., "Fuzzy Sets", Info and Control 8, 3, pp. 338-353, June 1968. (IVE)

228. Zadah, L. A., "Fuzzy Sets and Systems", Sympos. on System Theory, Poly. Institute, Brooklyn, New York, April 1965. (IVE)