AN INTRODUCTION TO THE HEURISTIC PROGRAMMING SYSTEM

David K. Jefferson

U. S. Naval Weapons Laboratory
Dahlgren, Virginia

ABSTRACT

The Heuristic Programming System is a language, designed but not yet implemented, for research in artificial intelligence It will provide facilities for creating, modifying, and destroying complex hierarchically structured objects and descriptions of objects A search operation will be provided to retrieve objects which are specified by arbitrarily complex descriptions Another search operation will construct the desired objects according to the specifications of previously created doscriptlve objects, this will be rather like a syntax-directed compiler for a continuously changing ambiguous language

A program for playing Go-Moku has been written using the System, the program features a highly efficient data structure, evaluation of feasible moves by alpha-beta minimax, and improvement of the move generation mechanism whenever the opponent makes a valuable but unexpected move

Key words and phrases problem-solving, heuristic, data structure, information retrieval, search, programming language, pattern recognition, description, learning, game playing, artificial intelligence, representation.

Overview of the System

The Heuristic Programmirng System consists of tho following major sections

1 The Class I anguage (CL), and
? The Problem Solving Fxecutive (PSF)

CL is the programming language in which the user describes the structure of a problem area it Is a considerably oxtended ALGOL which enables the user to croato, manipulate, ,end destroy sets and classes of sots Tho external form of the language has boon greatly influenced by LEAP, the Language for the Expression of Associative Procedures (Rovner and Feldman, 1967), which is an ALGOL based languago with facilities for manipulatinq sets and associative data However, both the purpose and internal structure of CL differ greatly from L EAP

Facilities for Representation

The basic. non-Al GOL structure In CL is the *set* Sets with common properties may be combined together into a *class,* the properties may have different values for different sets in the class Class members may be created or destroyed during a computation A class may be declared to be a *goal class* or a *descriptive class* tho PSE may construct members of a goal class from other sets, using members of a descriptive class as dox riptions of the goal class

A set may be accessed in various wiiys by name, by reference from a set contained within it, or by reference from a set within which it is contained If a set Is d mombor of a class, then its name is the name of the class followed by an index (i e , <i subscript) The user may insert a serios of integer-valued expressions between the name and index, indicating sunt lasses

Any set which is declared within a block is created upon entering the block, just as a local variable in Al GOL, sets are created empty The members of a class are created when entering a block if the class has a fixed cardinality (i e , number of members), otherwise members are created by the *create* prcx,edure Seis are destroyed when exiting from tho block in which they are declared class members may also be dostroyed by the *destroy* procedure The destruction of a set means that all properties of the set become undefined Further, *class members which contain a destroyed member are also destroyed,* all *references* from sets *contained* in the destroyed memt>er are deleted Numoric or Boolean properties may be defined in any of three ways a constant initial value may be assigned during the compilation of the block in which a set is defined, or an expression may be assigned at compilation time and then evaluated at the creation of the set, or the value may be assigned within the block, overriding any previously assigned value

Control Mchanisms

There are three basic types of control mechanisms in the Heuristic Programming System basic ALGOL (which will not be discussed further), the CL for (a search mechanism), and the PSE (a descriptive and constructive mechanism)

The CL for is a sophisticated search facility. The user writes a *template* which is basically the specification of the class name of an object, its structure (or various possible structures), and its properties The class name, and any of the class names appearing in the structural description, may be followed by a "defined" index or by an "undefined" variable The defined index refers to a specific class member, while the undefined variable refers to a momber which is to be found by the search When the desired member is found, its index is assigned to the variable. the member then can be manipulated by reference to its class name and index

Thus, the basic orientation of the Class Language is toward explicit *specification* and *search,* the user is spared the task of *choosing* and *identifying* objects 7 his tends to make CL code reflect the user's mental representation of objects, which facilitates coding and debugging

The CL for has been influenced by both LEAP (Rovner and Feldman, 1967) and SNOBOL (Farber, Griswold, and Polonsky, I9b4)

A CL for is used to search for objects which exist, i e , which have been constructed previously, a PSE statement may not only *search* for objects, but it may also *combine* objects, according to templates. In order to construction members of goal classes (after a goal object has been constructed, It can be manipulated by ordinary CL code) The PSE is, therefore, goal-directed, like the productions of the COGENT programming system (Reynolds, 1965) Unlike other systems, however, the PSE can construct more than one object, or it may not be able to construct any at all. Each invocation of the PSE must, therefore, specify the maximum amount of time to be spent, and whether one or many objects are desired

Facility for Improvamant

The most important control features of the Heuristic Programming System are Its methods for searching for specified objects, therefore, improvement consists of creating, destroying, and altering descriptions of objects Members of descriptive classes may be created and destroyed just as members of other types of classes, but the facilities for alteration are unique Basically, new descriptions aro produced in the following manner a goal class member is created, then a descriptive member is created which contains the goal class member, the descriptive member may be *edited* and then the descriptive member is *abstracted.*

Editing is a process by which a description is changed without changing the thing described, each set which is to be changed is copied, then the copy is changed All references to the original set within the description are changod into references to the copy Abstracting is a process of copying all sets, as above, and then changing the set names into undefined names Thus, the result is just a template

Tha Class Languaga

The detailed presentation of CL will assume a moderate knowledge of ALGOL and the metalinguistic formulae used to describe ALGOL. In particular, the revised ALGOL report (Naur *et al* , 1963) defines formally various metalinguistic variables which are used here, these aro, for the most part, self-explanatory. Both the metalinguistic variables u9ed to describe ALGOL and those U9ed to describe the additional constructs of CL are indicated in this Chapter by the brackets "(" and ")", generally there will be no difficulty in distinguishing ALGOL from the new constructs

Representation

Declaring sets and classes

A name may be declared to be a sot name by the declarator art or a class name by the declarators clan, goal date, or descriptive class. Sots are normally ordered, but may t>e declared to bo unordered, rank hi (high member first), or ranklo Ranking may be on the basis of any property of the members The members of a class may be declared to be unordered, rank hi or ranklo in the class declaration (the class. Itself, is always ordered) The cardinality of a class may be established at the time that the classs is declared, as In the following example.

class class [100]

This class is not empty initially, it consists of as many mombers as are declared, although these members need not have any defined properties

A class may be subdivided into subclasses, subsubclasses, and so on, to any desired extent This is indicated in the declaration as, e g..

d a n subclasses [1 0 ,5].

Here, there are 10 subclasses, each with five members.

If a cardinality Is not declared, then a class may have any number of members. This flexibility may be purchased at a cost in processing time, since an array structure is used for a class of fixed cardinality, while a list structure is necessary in the general case. A specific element of a class of fixed cardinality is accessed by "pointing" to it. otherwise the list of members must be searched to find it The search time is, however, quite small in many cases, because a record is kept of the location (on the list) of the most recently accessed member of each class, this greatly facilitates operations which sequence through the class or which repeatedly refer to the same member.

Note that if a class has a declared cardinality, then any member may be referred to (e.g.. may have values assigned to Its properties) without explicitly creating it. If a class has variable cardinality, on the other hand, each member must be explicitly created by a create statement before referring to It Thus, the two types of classes are quite different.

A class may be declared to have subclasses even though it has no fixed cardinality. Thus.

ctast manymember (5.1,

declares a class which has 5 subclasses, each with any number of members.

Parser ing properties

The standard properties structure and cover are automatically declared whenever a set or class is declared. The structure of a set is a list of the elements contained in it This is not the same as the set itself if A contains 6. and C contains the structure of A, then C contains B but not A. The cover of a set is the set of sets containing It The number of elements in either the structure or the cover may be declared. For example

clan triangle (integer structure -3, cover -5).

declares a class of triangles, each member of which contains at most three elements and is contained in at most five elements Such a declaration means that the pointers to the elements of the structure and cover may be placed in a contiguous block in each class member, rather than in lists. The value and cost properties are automatically declared for goal and descriptive classes.

Nonstandard properties must be declared explicitly, and may be assigned initial values. Thus

dees box (real length, width -2.. height - p + q).

specifies that each member of the "box" class has a length, a width (with value 2 when the member is created), and a height (with value p + q when the member is created, for the then current values of p and q).

Syntax of declarations

The syntax of declarations will now be given This defines not only the special CL constructs, but their relation to ALGOL as well

(declaration) . ■ (type declaration) I(array declaration) I
(switch declaration) |< procedure declaration) |(set
declaration > | (class declaration) | (goal declaration > |
(descriptive declaration) | (standard test)

Syntax of set declarations

(set name) - (identifier)
(property name) ■ (identifier)
(short property assignment) " (property name) - (arithmetic
expression) I (property name) .» (Boolean expression)
(property item) .- (property name) \ (short property assignment)
property list) . •= (property item) I (property item). (property list)
typed property list): - (type) (property list) | (type) (property
list), (typed property list)
(property declaration) :.- (empty) |((typed property list))
(set declaration Item) - (set name) |(property declaration)
(set declaration list) .:- (setdeclaration Item) | (set
declaration Item\ , (set declaration list)
(rank) « renkhi [ranklo
(modifier) -unordered j(empty) I (rank) |(rank)
by (variable)
(set declaration) - (modifier) set (set declaration list)

Syntax of class declarations

(basic class name) - (identifier)
(subclass and cardinality list) - (arithmetic expression) |
(arithmetic expression) . (subclass and cardinality list)
(subclass and cardinality declaration) - (empty) | [(subclass and
cardinality list)] | [(subclass and cardinality list) ,]
(class declaration item) = (basic class name) (subclass and
cardinality declaration) (property declaration)
(class declaration list) ■ (class declaration item) | (class
declaration item), (class declaration list)
(class declaration) "(modifier) class (class declaration
list)

Syntax of goal declarations

(goal class name) ■ (identifier)
(goal declaration item) ■ (goal class name) (subclass and
cardinality declaration) (property declaration)
(goal declaration list) - (goal declaration Item) I (goal
declaration Kern) ,(goal declaration list)
(goal declaration) . = (modifier) goal dass(goal declaration
list)

Syntax of descriptive declarations

(descriptor name) •- (identifier)
(descriptor Item) • (descriptor name) (subclass and cardinality
declaration) (property declaration)
(descriptor list) = (descriptor item) I (descriptor Item) ,
(descriptor list)
(descriptive declaration) - (modifier) descriptive dass
(descriptor list)
(class name) " (basic class name) I (goal class name) I (descriptor
name)

Syntax of procedure declarations

(procedure declaration) - procedure (procedure heading)
(procedure body) |(type) procedure (procedure
heading) (procedure body) I set procedure (procedure
heading)(procedure body)

Syntax of standard tests

(standard test) - standard test (property name) I (standard
test) . (property name)

This is a simple way to avoid writing the same condition in many (template)'s A (template) contains lists of class members and Boolean conditions on those members (a more complete discussion is given later) Each (property name) (which must be a Boolean variable) in the (standard test) is added as an additional condition to each member with that property, provided that no condition involving the (property name) is already present

Referring to members of sets and dassss

Members of a set or class are referred to by index and subclass indicators as, e g ,

set name. I

classname J

subclasses I.J

where I and J (here and in subsequent examples) may be arbitrary (primary)'s and set name is any set (in particular, it could be classname J).

Members of classes may be referred to by subclass Indicators and indexes which are either "defined" or "undefined", a period before a (primary) Indicates that It is defined, as in the .1 of

classname I

while a slash before a (simple variable) indicates that it is undefined, as in the /K of

classname/ K

The defined values are used to indicate a specific class member The undefined variables are used to indicate that an index or subclass indicator Is not known, as, for example, when searching for some member with specific properties or when creating a new class member When the desired member is found or created, its index and subclass indicators are assigned to the previously undefined variables. Similarly, a set name followed by a slash indicates an undefined set, to which structure is to be assigned, a set name not followed by a slash represents a previously defined set of elements. An asterisk followed by a period and an (unsigned integer) represents a variable to which structure has been assigned during a searching operation This will be discussed In more detail In the section on (template) 's.

Operations on sets

Sets are combined by means of the binary operators +, X and -. which are Interpreted as concatenation (with subsequent deletion of duplicated elements, if the result Is assigned to an unordered set), intersection, and subtraction. The precedence order is X first and last, as indicated In the syntax. Association is from the left in a sequence of identical operators, or may be indicated by parentheses. Note that brackets may be used to construct sets from lists of arbitrary expressions, hence, numeric or Boolean quantities may be put Into sets. Brackets are removed by the structure function so that. e.g..

F -[structuredA. B, C. D.]).[E]].

is equivalent to

F =[A,B,C,D,[E]],

Syntax of sets, expressions, and assignments

```
( expression)    = ( arithmetic expression ) | (Boolean expression ) |
    ( designational expression) | ( structural expression )
( assignment statement)   := ( left part list ) := (arithmetic expression) |
    ( left part list) := (Boolean expression ) | ( structural assignment )
( index )  = (primary )
( simple set )  = (set name ) | ( class name) | [ (structural expression ))
    | ( function designator ) | *. (unsigned Integer ) | ( simple set )
    ( index )
( set ) . = (simple set ) | [ (set list )] | empty
( set list )   = ( expression ) | ( expression ), ( set list )
( set factor)    = ( set ) | ( set ) X ( set factor )
( set term )  .= ( set factor ) | ( set factor )  + ( set term )
( set phrase)  ::= ( set term ) | ( set term ) - ( set phrase )
( structural expression ) . = ( set phrase ) | ( if clause ) ( set phrase )
    else (structural expression )
( structural assignment )  := ( simple set ) := ( structural expression) |
    ( procedure identifier ) .= (structural expression )
```

Creation off classmemoers

A class member may have values assigned to any of its properties by the statement which creates it. these values override values given in the class declaration. Values assigned to a member's structure are indicated as In the following.

create. (trlangle/L(area -10, llne.l, line.J, line.K)).

This indicates that a triangle is to be constructed which consists of (i.e.. whose structure is) the three lines, and whose area is defined to be 10 Any class member indicated In the structure may also be created, if its Index is undefined, as in the following

create. (intersecticWK(line/L(polnt.l.pointJ).line/M(polnt l.pointN))).

This process may be continued to any degree of nesting; during execution the effect is to create the leftmost member whose structure has already been created, then to repeat, until all members have been created. Any of the members being created may have values assigned to any of Its properties

Syntax of create

```
( undoflned index)    =/ (variable )
( defined member)   «  (class name ) I ( defined member ) ( Index )
( undefined member)    » ( defined member ) ( undefined index ) I
    ( undefined member ) ( undefined Index ) | ( undefined member )
    ( Index)
( newmomber)  ■ (undefined member) |( undefined member) ( (new
    description list) )
( description ) = ( set) I ( short property assignment) I
    ( new member )
( new description list) = ( description ) I ( description ),
    ( new description list )
( creation ) . •= create. ( (new member ))
```

Each of the (new member)'s is created by the single statement

Syntax of destroy

```
( destruction) . - destroy.( (set ))
```

Recall that sets which contain a destroyed set are also destroyed, and that references to a destroyed set from a contained set are deleted from the contained set's cover.

Manipulation off properties

A (property name) alone is used to Indicate a property where the set or class member is known by context, that is. within a (class declaration) or (new description list) or within a (template) (to be described later). Otherwise, the class member in parentheses follows the property name. Thus,

class box [1] (real area -2),
totalarea =2 Xarea (box.1).
area (box.1) ■ 5 X totalarea,

A single identifier may be the name for properties of many different classes (as, for example, cover and structure) since the class member is always made clear either explicitly or from context

Syntax of properties and variables

```
( variable)   - (simple variable ) |( subscripted variable ) I
    ( property variable )
( property variable )  .- ( property name ) |
    ( property name ) ( (simple set ))
```

Control

Templates

A (template) Is a description of a collection of sets It consists of a list of (set variables)'s (each one a (set name) followed by a slash, or an asterisk, an asterisk followed by a slash and an (unsigned integer), or an (undefined member)), (set) 's, and (Boolean expression)'s. The (set variable)'s are assigned values such that the structural conditions implied by the (set) 's and the Boolean conditions of the (Boolean expression) 's are satisfied For example,

triangle/Uarea - 10Alght. line.I, llne/K, line.J)

indicates a right triangle *(or* collection of right triangles) whose area Is 10 and which consists of line. I, a line (or collection of lines) whose index is unknown, and line J. No properties are specified for the lines (in particular, their structures are not specified).

If part of the structure of a set is irrelevant or unknown, an asterisk may be used In its place in a (template). The asterisk signifies that its place could be occupied by any string of symbols which represent structure (including the "empty" structure). Thus.

object/L (*, line.I, *)

specifies any "object" which contains llne.I One example of such an object is

object. 1 (line.1, triangle.2 (line.2, line.2. llne.l))

An asterisk followed by a slash and an (unsigned integer) is similar to a (set name) followed by a slash, the structure which is assigned to it may be referred to elsewhere by an asterisk followed by a period and the (unsigned integer) Thus.

for each line/K (V I , point.J. */2) do
begin create. (line/L (M , # 2)), destroy, (line K) and,

creates a collection of new lines not containing point.J.

The functions "and", "or", and "not" may be used within a (template) to indicate, respectively, structural conditions which must be simultaneously satisfied, or are alternatives, or are forbidden. For example,

Inside/L (or (Icircle/K. triangle/M]. [triangle/M. circle/K]))

specifies an object which consists of a circle "inside" a triangle, or a triangle "inside" a circle

Recall that Boolean conditions may be inserted into a (template) by means of a (standard test) . For example, if the following declarations are made

standard test active,
class triangle (real area. Boolean active),

then

trlangle/L (area = 10)

and

triangle/L (area = 10 AactIve)

specify the same collection of members, but

triangle/L (area = 10A"~lective)

specifies a disjoint collection

The CL for

The (CL for) is used to assign values to the (set variable) 's which appear in a (template) , structure is assigned to the (set name)'s which are followed by slashes and to the asterisks, and values are assigned to the undefined subclass indicators and Indexes of the (undefined member) 's Thus, the (CL for) Is essentially a sophisticated search procedure For example,

ffor each object/K (triangle/L (\ line.I,.),square/M (.. Ilne.I/))
do (statement)

searches for any object which consists of a triangle and a square with llne.l in common.

In the example above, the each specified that all possible assignments of values to L. K. and M were to be made which satisfied the (template). first could have been used instead, with the obvious significance If the first Is used, then clearly the results may depend upon the order in which assignments are made Note also that the (statement) could create, modify, or destroy class members, since the search locates one after another of the examples of the (template). the results of the for each may also depend upon this order

A second type of (CL for) is used to assign each element, one after the other, (or the first element, or the last element) from a specified (set) to a (set name) The process is simply one of renaming sets. Thus, for example, the statement

```
    for seen A in [B. C. D. E] do A -A+F,
```

is equivalent to the sequence of statements

```
    B =B+F.C =C+F.D =D+F.E =E+F.
```

In both of the forms of the (CL for >, as in the ALGOL (for statement >, the (statement) following the do is not executed at ail, if the conditions of the (CL for) cannot be satisfied Thus, the (CL for) may be used to determine whether certain class members exist

Syntax of the for statement

```
⟨ for statement ⟩ := ⟨ for clause ⟩⟨ statement ⟩|⟨ label ⟩
    ⟨ for statement ⟩|⟨ CL for ⟩⟨ statement ⟩
⟨ set variable ⟩ := ⟨undefined member⟩|⟨ set
    name ⟩/|*|*/⟨unsigned Integer⟩
⟨ basic structure ⟩ := ⟨set variable ⟩|⟨set⟩
⟨ template ⟩ = ⟨ basic structure⟩ | ⟨basic structure ⟩
    (⟨condition list ⟩)
⟨ condition item ⟩ = ⟨ Boolean expression ⟩ | ⟨template ⟩
⟨ condition list ⟩ = ⟨ condition item ⟩ | ⟨ condition item ⟩.
    ⟨ condition list ⟩
⟨ for adjective ⟩ : = each | first
⟨ set adjective ⟩ := each |first |last
⟨ CL for⟩ := for ⟨for adjective ⟩
    ⟨ template⟩ do |for ⟨set adjective ⟩
    ⟨ set name ⟩ in ⟨set⟩ do
```

The Problem Solving Executive

An Invocation of the PSE is. as discussed earlier, similar to a ⟨ CL for⟩ „ with the important addition that the PSE can not only search for specified objects, but may also construct them. This construction is limited to the construction of members of goal classes from other class members. Thus, members of non-goal classes must be created by declaration or explicit create statements, this is necessary since the non-goal classes are the objects of the problem and are manipulated by the rules of the problem, rather than by the PSE. The goal classes, on the other hand, represent hypotheses about the problem, and hence are subject only to rules defined by the problem solver

The desired goal members are described to the PSE by means of a (template) and/or members of one or more descriptive classes. For example,

```
    for first subgoal/I (object J, object K) during
     time do (statement)
```

specifies that a "subgoal" is to be constructed from the two "objects" if they exist and can be found within the time limit, "time" In this case, the (template) completely describes the desired subgoal. so no descriptive member is referenced. On the other hand,

```
    for each subgoal/K (object J. .) during time
    do ( statement)
```

incompletely specifies the desired subgoals the structure implied by both the (template) and by some descriptive member must be satisfied by each subgoal Finally,

```
    for each subgoal/K during time do (statement )
```

and

```
    for each subgoal/Ktvalue >bestval) during
    time do ( statement )
```

do not make any structural restrictions on the desired subgoals. structure must be supplied by descriptions which were created earlier

If it is desirable that the goals be described by members of a specific class, or by a specific member, this can be indicated in the (template) For example,

```
    for each description/K (subgoal/L) during time do
   \ statement /
```

or

```
    for each description.! (subgoel/L) during time
     do (statement)
```

In each of the above examples, time is initially set to the time allowed, whenever control leaves the PSE, time is equal to the remaining amount of time. Thus, it Is rather easy to devise quite complex ways of allocating search effort. For example, the following code searches for "moves" which will achieve "subgoals" of increasing value After N moves have been found and placed In moveset, the remaining time is devoted to finding "traps" (presumably very high-value goals) This might be a section of a game player

```
    moveset = empty, bestval =0, M =0,
    for each subgoal/K (move/L,*,value>bestval)
    during time do
    begin moveset := [move L] + moveset,
        bestval = value (subgoal.K),
        M :=M+1,
        if M = N then go to exit end,
    exit  for each trap/K (move/L, *, value >
    bestval) during time do
    begin moveset =[move L] + moveset.
        bestval := value (trap K)
       -

    end;
```

```
Syı ⟨PSE statement ⟩ := for ⟨ for adjective ⟩ ⟨template ⟩
      during ⟨ variable ⟩ do
      ⟨statement ⟩
```

Improvement

The basic operations of Improvement in the Heuristic Programming System are the construction, destruction, and alteration of members of descriptive classes The construction of a new descriptive member consists of the following steps first, recognition of a new goal member, second, creation of a descriptive member whose structure consists of this goal, third, editing of the descriptive member to alter structure *or* properties, fourth, abstraction of the descriptive member

Editing, as noted earlier, is a process by which a description is changed without changing the thing described An (edit statement) consists of the name of the description and either a single (replacement) or a block of (replacement)'s Each (replacement) either assigns a new value to each instance In the description of a given property, or replaces each instance of a given (template) In the description by new structure and property values A (template) is deleted from a description If It Is replaced by null Each (replacement) replaces part of the description by a reference to something which is strictly local to the description. In this way arbitrary replacements may be made without any effect upon any "external" objects, but It is still possible to refer to "local" objects by the names of the corresponding external objects.

Abstraction is a process of changing a (possibly edited) description into an "abstract" object, in which all indexes are undefined All pointers to external objects are replaced by pointers to local blocks, which represent the objects and all their properties (including subclass indicators) Essentially, this is just the process of changing a class member into a (template) which contains only (undefined member)'s.

The improvement process will be illustrated by a somewhat detailed example, the generation of a subgoal description in a tictac-toe program Although the application is trivial, the idea behind this example is quite powerful

Assume that the data structures of the program *are* "squares", "lines", "subgoals", and "descriptions" Each square has an integer-valued property called "occupant" with values "X", "0" or "unoccupied". Each line has *an* integer-valued property called "occupant" with values "X", "0", "unoccupied", or "blocked", and an integer-valued property "number" which may have any value from zero to three (zero if the line is blocked, otherwise the number of occupants) Each line consists of three squares. A description of a subgoal consists of an unoccupied square, a list of lines and the squares which they contain, and the properties of the lines and squares. Each subgoal has two properties, its "side" ("X" or "O") and Its "subgoelvalue", with a value of $1/2^n$ for some n. The Interpretation is that If the present configuration should contain the specified lines and squares and if the player with the proper side should occupy the square, then the resulting configuration would lead to three in a line in n moves or less regardless of the moves made by the opponent (although if he has a more valuable subgoal, he might be able to achieve three In a line first, and therefore win).

Assume now that the opponent, X, has just moved to square.I and that his move has created two subgoals, subgoal.J and subgoal.K, with values 1/2%nd $1/2^n$, which cannot be simultaneously blocked Evidently his previous *move* occupied the square of a subgoal with subgoelvalue equal to the minimum of $1/2^{m+1}$ and $1/2^{n+1}$. The problem is to create a description of this subgoal.

The first step has already been done: the relevant objects, square.I, subgoal.J, and subgoal K. have been found. The second step is to creato a description of these objects

```
    create. (description/L (subgoal/M (subgoalvalue. =If subgoalvalue
        (subgoal. I )> subgoal value (subgoal.J) then subgoalvalue (subgoal J)/2.
        else subgoalvalue (subgoal.I)/2., side =X. square.I, structure
        (subgoal.J). structure (subgoal K)))).
```

(Note the use of "structure" the subgoal will consist of lines and squares, not of other subgoals.) This object is a description of the situation after, rather than before, trie move has been made. Thus, the occupant of square.I. and its effects, must be removed

```
    edit description.L do
    begin occupant (square.I) -unoccupied, line/N (V1. square I.V2) -
        line.N (number -number (line.NM, * . 1 , square.I, *.2)
    end;
```

Now the description is abstracted

```
    abstract (description.L),
```

This is a complete description of a subgoal which the opponent can achieve. A subgoal which the program can achieve is produced by the following

```
    create.(descrlption/M(structure (description.L))).
    •dh description.M do
    begin side (subgoal/N) -O.
        occupant (line/N).-If occupant (line.N) -X then O else If
            occupant (line N) - O then X else occupant (line.N),
        occupant (square/N) - If occupant (square.N)-X than O else if
            occupant (square.N) - O than X else unoccupied
    end
```

Syntax of editing

```
    (member)  - ( defined member) I( undefined member )
    ( property replacement ) .=( property name )( (member))-
        (Boolean expression ) K property name )( ( member) ):=
        (arithmetic expression )
    ( structural replacement) -(template) -null I
        (template ) - (new description list )
    ( replacement) '- (property replacement) |( structural replacement )
    (compound replacement) - ( replacement) I( replacement ).
        (compound replacement)
    ( edit block ) - ( replacement) (begin (compound
       replacement) end
    (descriptive member ) - (descriptor name )( Index ) I( descriptive
       member ) ( Index )
    ( edit statement) * edit ( descriptive member )
       do (edit block )
```

Syntax of statements

```
    (statement ) - ( unconditional statement ) I( conditional statement )I
        (for statement) I( PSE statement ) I( creation ) I( destruction ) I
        (edit statement )
```

Conclusion

The foregoing is part of a much larger paper which includes a proposed implementation and a sample program which plays the game of Go-Moku (Jefferson, 1969). The Go-Moku program has the following interesting features positions are represented efficiently and. to the human programmer, very naturally, the distribution of playing time within the program is easily and flexibly controlled Improvement of move generation and evaluation is accomplished by creating a description of each of the opponent's moves which unexpectedly leads to a valuable position, alpha-beta minlmax is easily and efficiently implemented, greatly increasing the playing ability of the program. The facilities of the System made the programming reasonably simple and straightforward, this was due in part to the simplicity of construction of complex objects and descriptions of objects, and in part to the simplicity of searching (by means of the PSE) for instances of complex objects. The ease with which operations on one object may be propagated to other (dependent) objects contributed greatly to the efficiency of the program.

Possible areas of application of the System include other games such as chess or checkers, pattern recognition, and various allocation and scheduling problems of operations research. The System is unsuitable for such symbol-manipulation problems as theorem proving and symbolic Integration, other programming systems, such as LISP 1.5 (McCarthy, et al.. 1965) or SNOBOL (Farber. Griswold. and Polonsky. 1964) are much to be preferred. However, these systems are awkward and inefficient in the problem areas Involving the construction and recognition of complex, multi-dimensional, hierarchical objects, for which the Heuristic Programming System is most suitable.

List of References

Farber, D , Griswold. R . and Polonsky, I., 1964 "SNOBOL, a string manipulation language." Journal of the Aaociatlon for Computing Machinery, 11 (2) 21-30.

Jefferson, D., 1969 A Heuristic Programming System, thesis to be submitted to the University of Michigan. Ann Arbor

McCarthy, J. et al., 1965 LISP 1.5 Programmer's Manual. Cambridge The MIT Press.

Naur, P., et al., 1963 "Revised report on the algorithmic language ALGOL 60," Communications of the Association for Computing Machinery, 6(1) 117

Reynolds, J., 1965. "An introduction to the COGENT programming system." Proc. 20th National Conference of the Association for Computing Machinery: 422-436.

Rovner, P., and Feldman, J. A., 1967 "An associative processing system for conventional digital computers." Technical Note 1967-19, Lincoln Laboratory. Massachusetts Institute of Technology, Lexington, Mass