Chapter 3 Syntactic Analysis of Visual Sentences

1. A Visual Language Compiler

In Chapter 1, we introduced elements of the formal theory of visual languages. The main notion can be stated as follows: *a visual language can be designed, by specifying a system of generalized icons.*

In this section and the following sections, we describe the design of a visual language compiler, called the *SIL-ICON Compiler* (Syntactic, Interactive, Learning, ICON-oriented-system Compiler), which is a software system for the specification, interpretation, prototyping, and generation of icon-oriented systems. The SIL-ICON Compiler has the following characteristics:

Syntactic Specification: It uses a formal grammar G as a specification of iconic systems.

Interactive Usage: It accepts user's specifications interactively. The user can access icons defined in the Icon Dictionary (ID), and use operators in the Operator Dictionary (OD), to create any icon in the icon world.

Learning Capabilities: The user can modify ID, OD and G, to expand or specialize the domain of applications.

Another way of describing our approach, is that the SIL-ICON Compiler accepts *Symbolic Images* and *Logical ICONs* as inputs, to allow the user to design a visual language or an icon-oriented user interface.

The system diagram of the SIL-ICON Compiler is illustrated in Figure 1.8. It includes the following parts:

- **G** *Icon System G* is a formal, syntactic specification of the iconic system.
- **ID** *Icon Dictionary* is a database corresponding to a set of elementary icons.
- **OD** *Operator Dictionary* is a set of generic operators which can be applied to icons to create new complex icons.
- **ETAG** *Extended Task Action Grammar* provides a formal method for describing the user's conceptualization about tasks to be performed (This part will not be discussed in this chapter).
- **ICIN** *Icon Interpreter* is the heart of the system. An input iconic sentence (visual sentence) s, which is a spatial arrangement of icons (i.e. a complex icon in the icon world), is interpreted by the Icon Interpreter to arrive at a visual concept Vs. The Icon Interpreter may be domain-specific, so that it will perform the specific task corresponding to the visual concept. On the other hand, a general-purpose Icon Interpreter should have a formal knowledge structure in its knowledge-base. The extended task grammar ETAG specifies a user's conceptualization of an application domain, and the correspondence between visual concepts and corresponding tasks [TAUBER86, 87]. Therefore, with the formal specification (G, ID, OD, ETAG), the Icon Interpreter can simulate an icon-oriented user interface.

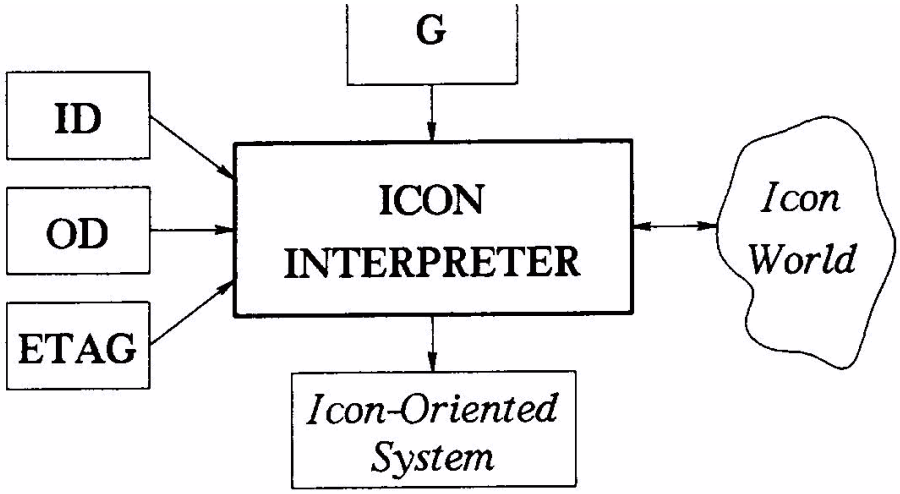


Figure 1.8: The System Diagram of SIL-ICON Compiler

As a compiler, the SIL-ICON Compiler accepts (G, ID, OD), the user's formal specification of an icon-oriented system as its input, then generates the *realized icon-oriented system* as its output.

Let B denote the set of all elementary icons of an iconic system G. Let OD denote the set of operators. Then an *icon world* is formally defined as the closure of (OD,B), i.e., all the composite icons constructed from B using operators from OD. An icon world can theoretically have infinitely many icons, even though most of them will not be useful for a specific application domain. An icon set selected by the user for an application domain is a subset of icon world, which is called an *icon-oriented system.*

Therefore, an icon-oriented system is an application-oriented subset of its icon world. Each icon in an iconic system is defined, or characterized, by its semantic descriptions. Such descriptions can be formalized as the evaluation rules associated with the icon rules. The icon set with evaluation rules, ER, is defined as,

$$ER = \{ (x, X_m, X_i, e_x) : \quad x ::= (X_m, X_i) \in R \}$$

where e_x is the evaluation rule for icon x. The evaluation rules are names of executable procedures. The user can provide the executable procedures, in which case these procedures will be used as the evaluation rules. If such procedures are not provided, then the SIL-ICON Compiler will perform icon interpretation according to (G, ID, OD).

2. The Icon Dictionary ID

An icon x(Xi,Xm) in its external representation is stored in the icon dictionary ID. The name `x' is an inherited or arbitrarily assigned name of an icon by which a specification (Xi,Xm) of the icon can be found in ID.

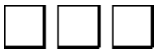
2.1. Specification of Physical Part of an Icon

An icon image can be a picture, a sketch, or a symbol. It can be regarded as composed from different primitive patterns. Therefore, the physical part of an icon can be described by a picture grammar. For example, the following picture grammar PG is defined for the description of the Heidelberg Icon Set (see Appendix 1).

```
PG(V, T, Y, P)    where

 V = {Y, <primitive>, <i-op>}
  is a set of nonterminals;
 T = {gap, rectangle,...,cross, (, ), +, &. ^}
  is a set of terminals;
 Y is the starting symbol;
 P is a set of rules:
  Y --> ( Y  <i-op>  Y ) |  <primitive>
  <primitive> --> gap | rectangle |
    d_rectangle | square | d_square |
    diagonal | r_arrow | d_arrow |
    u_arrow | b_arrow | cross
                <i-op> --> + | & | ^
```

In the picture grammar PG, the syntactic category <primitive> stands for different primitive patterns, and <i-op> stands for spatial image operators such as & (spatial overlay), + (horizontal concatenation), and ^ (vertical concatenation).

With this picture grammar PG, the icon for this string taken from the Heidelberg Icon Set



is described as *(square + (square + square))* or *((square + square) + square).*

2.2. Specification of Logical Part of an Icon

For the specification of the logical part of an icon, a *conceptual representation* is used. In this chapter, the specification of the meaning of the example icons defines the conceptual knowledge conveyed by these icons to the user. However, this specification also can be regarded as a high level conceptual description of the tasks performed by the icon-oriented system. For the specification of concepts and their conceptual structure, we will use conceptual graphs [SOWA84]. A *conceptual graph CG* is a finite, connected, bipartite graph. One node type represents concepts, the other conceptual relations. A conceptual relation has one or more arcs linked to concepts so a relation can be monadic, dyadic, ..., or n-adic. The smallest CG is a single concept, but a single relation alone may not form a CG. In this chapter, we use the following notation to represent a CG:

```
[CONCEPT] -> (RELATION) -> [CONCEPT] , or

[CONCEPT] -- (RELATION) -> [CONCEPT]

 (RELATION) -> [CONCEPT]  (for multiple arcs)
```

The CG used in the chapter is a nested graph: each concept can be again represented by a CG. Therefore a complex CG can be represented by a top level graph and subgraphs to be used to replace concepts in the graphs.

A concept is denoted by [] and the name used inside the brackets is the *type* of the concept. For example, [ICON] is a concept representing an unspecified individual of type ICON, i.e., a *generic concept.* [ICON:*] is an equivalent notation, where * is called the *generic marker*. [ICON:*x] stands for an arbitrary individual of type ICON. An *individual marker* is used when a particular individual of a concept is denoted. For example, [ICON: #1] is an *individual concept.*

For an icon world, we define S_C as the set of basic concepts, S_R as the set of conceptual relations, and I as the set of individual markers. For each concept c in S_C, the following functions are defined:

REFERENT(c) returns either the generic marker * or an individual marker #n in I, and

TYPE(c) returns a type label t.

Types are related together as specified by a *type hierarchy* which is defined by a partial ordering [SOWA84]. In this chapter, the conceptual framework used for the example icons is defined by the types OBJECT, PLACE, PATH, STATE, and EVENT. Subtypes of OBJECT are, for example, LINE and STRING. Subtypes of PLACE include ROW, and subtypes of EVENT include INSERT_LINE. Figure 1.9 illustrates this hierarchy of types.

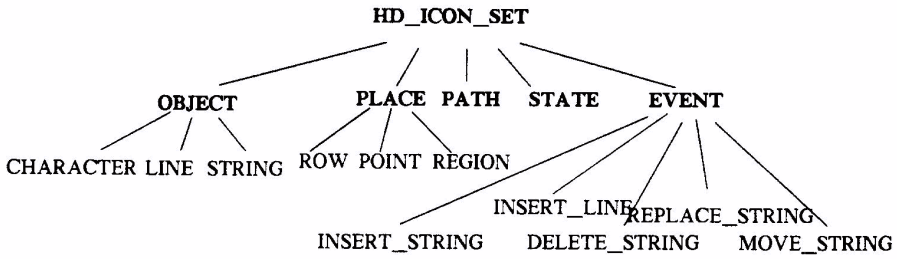


Figure 1.9: Type Hierarchy of the Heidelberg Icon Set

Sometimes we will specify a concept and its supertype. For example, [OBJECT = LINE] is a generic concept [LINE] which has [OBJECT] as its supertype.

2.3. Structure of ID

The Icon Dictionary ID is a database consisting of elementary icons. Each entry of ID stores an icon and is defined in the following format:

```
   ENTRY:=NAME,SKETCH{;SKETCH},DESC,TYPE,[EVAL],
 CG{;TYPE,[EVAL],CG},[ATTRIBUTES].
```

where the braces {} denote repetitions of an item. The meaning of the items is as follows:

- *** NAME ----**unique identifier for an icon or a type label.
- *** SKETCH --**a pattern string described by a picture grammar.
- *** DESC ----**a concise description of meaning of an icon for inclusion in the manual.
- *** TYPE ----**a type label indicating object icon, process icon, etc.
- *** EVAL ----**an optional item used by the Icon Interpreter to generate the action part of this icon, typically a procedure name.
- *** CG ------**a conceptual description of icon semantics generated by conceptual analysis.
- *** ATTRIBUTES ---**optional attributes of an icon.

In the following example of an icon-oriented text editor using the Heidelberg Icon Set, the optional items EVAL and ATTRIBUTES are not used. A formal grammar IG describing ID entries is given in Appendix 3.

2.4. Examples for the Definition of ID

In order to illustrate how the SIL-ICON Compiler works, the Heidelberg Icon Set (HD Icon Set) will be used as an example. From this set, we will consider five simple text editing operations: *insert line, insert string, delete string, replace string,* and *move string*. The Heidelberg Icon Set was constructed for psychological experiments on visually supported human concept building in text editing. The icons are used as command symbols for a menu (instead of verbal symbols) in a special text editor [KEPPEL&ROHR84], [ROHR&KEPPEL84], [ROHR86].

Figure 1.10 shows the five HD icons: *insert line, insert string, delete string, move string,* and *replace string.*

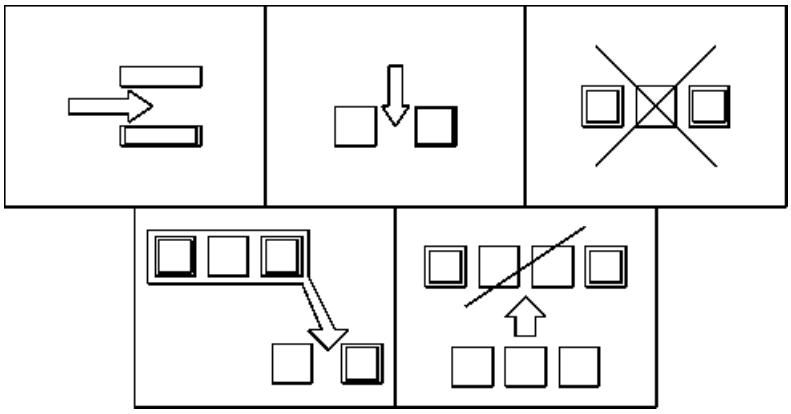


Figure 1.10: Icons from the Heidelberg Icon Set

Each of the five HD icons is built from some elementary icons. The icon for *insert line,* for example, consists of a horizontal right pointing arrow, an icon for a line, and an icon indicating a line marked as *that line*. The meaning of this complex icon is the insertion of a line to the position before *that line*.

Only elementary icons are stored in ID. Therefore, in ID we do not have the five HD icons. We will define the elementary icons the HD icons are built from: the rectangle, the square, the filled rectangle, the filled square, the cross, the diagonal, the arrow, and the arrow with a box. For all these icons, the physical part is defined in the SKETCH item, in terms of the grammar PG. The logical part is described by a CG.

As elementary object icons we have icons for line and character. A rectangle symbolizes the former, and a square the latter. The diagonal line signifies a place where an indicated object may be exchanged by another object. Arrow and cross are elementary process icons. The arrow signifies a movement of an object to a place. The cross signifies the deletion of an object.

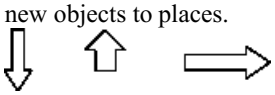
The *replace icon* signifies the replacement of a string by another string. The elementary icons used in the replacement icon are: the arrow, the diagonal, and squares. The meaning expressed by these elementary icons is as follows:

- -A string composed of characters (signified by a sequence of three squares) is moved to
- -a place defined by a marked string (signified by filled squares and normal squares) which is ready for
- -exchange (signified by the diagonal).

The following two examples of ID entries illustrate how the ID is described. A full description of the five complex HD icons, the intermediary icons, and the elementary icons, is given in Appendix 1.

```
CHARACTER, square,"a character", OBJECT,,
[OBJECT = CHARACTER].
```

This entry describes the elementary icon signifying an arbitrary character. The next icon describes the arrow as signifying movements of new objects to places.



```
ARROW, d_arrow; u_arrow; r_arrow,
"movement of new objects", EVENT,,
[EVENT = INSERTION] -- (THEME) -> [OBJECT]
(event.GO) -> [PATH : P.1] -> (place.TO) -> [PLACE].
```

The meaning of an elementary icon is described by a conceptual graph (CG). The definition of an arrow shows that different icon images can sometimes signify the same meaning. We also can have different meanings for the same icon image. This is the problem of icon purity, which was discussed in Section 8.

The five HD icons we consider are built from the elementary icons defined in ID. The synthesis of these five icons is based upon operators in OD, which will be explained in the next section.

3. The Operator Dictionary OD

Iconic operators are special icons which carry certain meanings based on actions to be performed. They may be represented by conceptual graphs. However, it is more convenient to represent them directly as procedures. The basic actions performed by iconic operators include:

```
- constructing the image of a resultant icon;
- constructing the conceptual graph of a resultant icon;
- synthesizing descriptions;
- determining the type label of resultant icon;
- determining the common attributes of icons; and
- optionally performing evaluation.
```

The following functions perform various transformations and mappings on an iconic representation. These functions are useful, when we define the iconic operators.

```
- find(x)  returns the specification (Xi,Xm) of icon x.
- flp(x)   returns the logical part Xm of icon x.
- fpp(x)   returns the physical part Xi of icon x.
- pmatch(Xi) returns an icon name x whose icon image is Xi.
- lmatch(Xm) returns an icon name x whose meaning is Xm.
- mat(Xm)   returns Xi of icon x (materialization).
- dmat(Xi)  returns Xm of icon x (dematerialization).
```

We now describe the most important operations to be performed by the iconic operators.

The following procedure constructs the image (physical part) of a resultant icon from input icons x and y, using image operator i-op.

```
procedure  ppart(x, y, i-op)
  icon x,y;
  char i-op; /* i-op is an image operator such as '*' , '+' or '^' */
  begin
     sketch = image-operation( i-op, fpp(x), fpp(y) );
     draw(sketch); /* draw the sketch on screen */
  end;
```

In the above procedure, i-op corresponds to the image operator i-op in the picture grammar. The procedure draw(sketch) parses the pattern string using the picture grammar and draws the sketch.

The following procedure constructs the meaning (logical part) of a resultant icon as a new conceptual graph, from input icons x and y.

```
procedure  lpart(x, y)
  icon x, y;
  begin
    restrict (x, y);
    join(x, y);
    simplify(x, y);
  end;
```

There are four generic procedures to perform operations upon the conceptual graphs: copy, restrict, join and simplify [SOWA84]. The generic procedures are explained below. In what follows, u and v are input CGs, and w is the resultant CG.

copy(u): The CG w is a copy of another CG u.

```
restrict(u,v): For any common concept c in CG u and CG v, type(c) is replaced by its
                  subtype.  If c is generic, its referent is changed to an individual
                  marker. These changes are permitted only if referent(c) conforms
                  to type(c) before and after the changes.
    join(u,v): If a concept c in CG u is identical to a concept d in CG v, then
                  a new CG w is constructed by deleting d and linking c to
                  all conceptual relations (arcs) that were previously linked to d.
  simplify(u): If conceptual relations r and s in CG u are identical,
                  then one of them is deleted from u together with all the
                  conceptual relations (arcs) previously linked to it.
```

To illustrate how we define iconic operators, procedures for the iconic operators VER and HOR are described below, where VER corresponds to the image operator '^', and HOR corresponds to the image operator '+'.

```
procedure ver(x,y)
 icon x,y;
  begin
   ppart(x, y);
   prep(x,y,"VER");
   lpart(x, y);
  end;

procedure hor(x,y)
 icon x,y;
  begin
   ppart(x, y);
   prep(x,y,"HOR");
   lpart(x, y);
  end;
```

In the above, the procedure ppart(x,y) constructs the physical image of an icon, and the procedure prep(x,y,op) synthesizes icon descriptions, determines type label of resultant icon, determines the common attributes of icons, and performs any domain-specific evaluation. If the conceptual graph of an icon has multiple sub-entries (i.e. the icon is semantically ambiguous), prep(x,y,op) may also have domain-specific rules for selecting a unique sub-entry (i.e. it disambiguates the icon meaning). Finally, the procedure lpart(x,y) constructs the logical meaning of an icon.

The above described procedures serve as examples illustrating iconic operators. An iconic operator can be thought of as an imageless icon, that is, the physical part of such an icon is null. An iconic operator is used to represent certain relationships between icons. These relationships could be either *physical relationships* or *logical relationships,* or both. At the expense of clarity, a procedure implementing an iconic operator can also be represented by a conceptual graph. For example, the iconic operator HOR can be represented by a CG with

formal parameters x and y:

```
   x,y [HOR]---

            (PHYSICALPART)--->[HORi]---

                                  (L-PARM)--->[ICON:x]

                                  (R-PARM)--->[ICON:y]

            (LOGICALPART)--->[HORm]---

                                  (L-PARM)--->[ICON:x]

                                  (R-PARM)--->[ICON:y]
```

4. An Example

To illustrate how the Icon Interpreter works, an example of the semantic analysis of a complex icon will be presented. The following icon rule set G1 describes a part of the Heibelberg icon set (see Appendix 2 for other icon rules of the HD Icon Set):

```
   G1:

Insert_line ::= ( {HORm, Right_arrow, On_row},

    {HORi, Right_arrow, On_row},

    "Insert_line" );


Right_arrow ::= ( ARROW, r_arrow, "Generic_Arrow" );


On_row ::= ( {VERm, Line, Gap, Marked_line},

    {VERi, Line, Gap, Marked_line},

    nil );


Line ::= ( LINE, rectangle, nil );


Marked_line ::= ( MARKED_LINE, d_rectangle,

 "Line_Pointer" );


Gap ::= ( GAP, gap, nil );
```

Since the icons ARROW, LINE, MARKED_LINE are all elementary icons, the Icon Interpreter can obtain the descriptions of the elementary icons from ID while it parses an input iconic sentence using G1. At the same time the structure of the complex icon in question also can be determined.

In the following, we describe how the iconic operators HOR and VER can be applied to produce the icon INSERT_LINE. In order to produce INSERT_LINE, the complex icon ON_ROW is to be constructed first. The construction of the image part of a complex icon using HORi (corresponding to the spatial operator +) or VERi (corresponding to the spatial operator ^) is straight forward. Therefore, we will only demonstrate the construction of the meaning part of the ON_ROW icon.

For the production of the conceptual graph for ON_ROW, we merge the involved CGs at those nodes where the corresponding concepts in both CGs satisfy matching conditions as defined by the type hierarchy (see Section 11). Given the iconic sentence "VER, LINE, GAP, MARKED_LINE", we construct the parsed expression VER(LINE,VER(GAP,MARKED_LINE)), where the logical operator VER combines icon LINE with the resultant icon of GAP and MARKED_LINE to form the icon ON_ROW.

First we consider the operation VER (GAP, MARKED_LINE). As we see in Appendix 1.1, GAP is a complex icon which defines a subtype of the concept [PLACE] defined by the place.FUNCTION BEFORE and an unspecified subtype of [OBJECT]. Now let us take (1), the CG description of the icon GAP, and (2), the CG description for icon MARKED_LINE, as an example to illustrate the merging of CGs. It should be noted that [LINE = MARKED-LINE] is the only [OBJECT] which may be used for the definition of the subtype [PLACE = ROW] to the type [PLACE = BETWEEN].

```
[PLACE = BETWEEN] -> (BEFORE) -> [OBJECT]         --- (1)

[OBJECT = LINE = MARKED-LINE]                     --- (2)
```

As we mentioned in Section 11, the iconic operator recognizes the equivalence among the concepts in both CGs so that equivalent concepts can be joined together. In the above example, the concept [OBJECT = LINE = MARKED-LINE] in icon LINE can be joined with concept [OBJECT] in icon GAP, and the result is:

```
[PLACE = ROW] -> (BEFORE) -> [OBJECT = LINE = MARKED-LINE]    --- (3)
```

Then the operator combines the CG (3) with the CG for LINE which is

[OBJECT = LINE] --- (4)

Since (4) does not restrict (3), with (3) we have obtained the final CG for ON_ROW.

Similarily, the CG description of icon INSERT_LINE can be derived from the complex icon ON_ROW and the icon ARROW. The icon INSERT_LINE is produced by HOR(ARROW, ON_ROW). Its CG will result from joining the conceptual graph for ARROW which is

```
[EVENT = INSERTION] -- (THEME) -> [OBJECT]
                       (event.GO) -> [PATH =P.1] -> (path.TO) -> [PLACE]
```

with the conceptual graphs

[OBJECT = LINE] and [PLACE = ROW].

The final CG for INSERT_LINE is then

```
[EVENT = INSERT_LINE] -- (THEME) -> [OBJECT = LINE]
                         (event.GO) -> [PATH = P.1.1] -> (path.TO) -> [PLACE = ROW]
```

To facilitate human understanding, the above CG can be translated into a description in natural language. Another interesting use of the CG, is to compare it with user's conceptual understanding of the task, so that we can find out whether the meaning of the complex icon is consistent with user's conceptual understanding.

5. Implementation of the Visual Language Compiler

The SIL-ICON Compiler is an interactive system. The main function can be summarized by the following procedure:

```
procedure SIL-ICON_COMPILER()
begin
 initialize ID and OD by loading the files;
 repeat
   display the current status;
   prompt the user to select one of
     the following functions:
  a. re-initialize ID or OD;
  b. interactively input/modify ID or OD;
  c. load an edited G;
  d. interactively input/modify G;
  e. invoke Iconic_Sentence_Parser;
  f. invoke Icon_Interpreter;
  g. invoke Generator;
  h. quit;
 until quit;
end SIL-ICON_COMPILER;
```

As an interactive system, the SIL-ICON Compiler provides a user oriented interface. This feature is supported by function **a** through **d.** These functions allows the user to interactively modify G, ID and OD. The Iconic Sentence Parser and the Icon Interpreter also allow

some user interaction.

The Iconic Sentence Parser allows the user to interactively create an iconic sentence by combining predefined icons. The iconic sentence is then parsed. After semantic interpretation, a natural language description of its meaning is generated.

The Icon Interpreter is the heart of SIL-ICON Compiler. It also generates icons according to ID and OD, but the Icon Interpreter generates icons under the control of the formal icon grammar G.

The syntax for the visual language is specified by G. The working environment of the SIL-ICON Compiler includes G, ID and OD. The iconic sentence s entered by user will be parsed according to G, and then the resultant complex icon structure will be given in the form of CG.

The Icon Interpreter is described by the following procedure:

```
Icon_Interpreter()
begin
     Parse an input iconic sentence using G;
     Invoke iconic operators in OD to evaluate the parsed
       iconic sentence and construct conceptual graph CG;
end Icon_Interpreter
```

The evaluation of a parsed iconic sentence essentially follows the recursive evaluation strategy described as the EVAL procedure in Section 6.

Finally, the Generator generates a realized icon-oriented system as its output, according to the evaluation rules associated with the icon rules, so that the realized icon-oriented system can be reused by the user. Therefore, the SIL-ICON Compiler serves the triple functions of icon world navigation, icon interpretation, and icon-oriented system generation.

An example of the SIL-ICON compiler is given in Figure 1.11. Figure 1.11(a) illustrates the screen layout for the iconic sentence parser. In the top row, various functions can be selected, to draw an icon, copy an icon, parse the iconic sentence, restore a previously saved icon, move the icon, rotate the icon, perform scale changes of icons, clear the screen, load icons from icon library, enter text, and exit from parser. In the bottom row, the predefined icons are displayed.

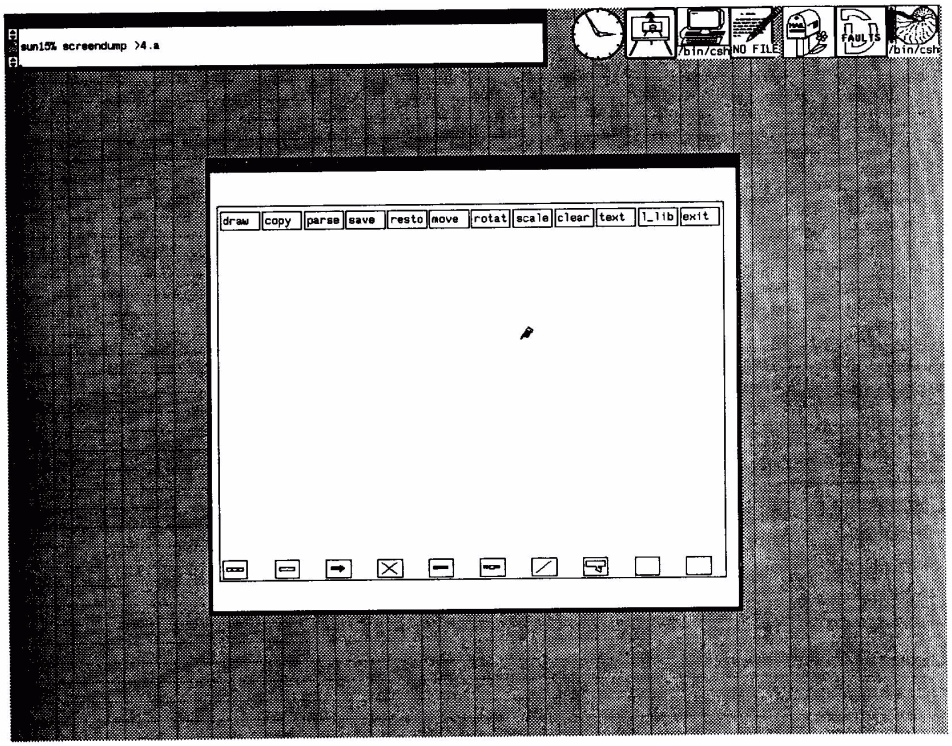
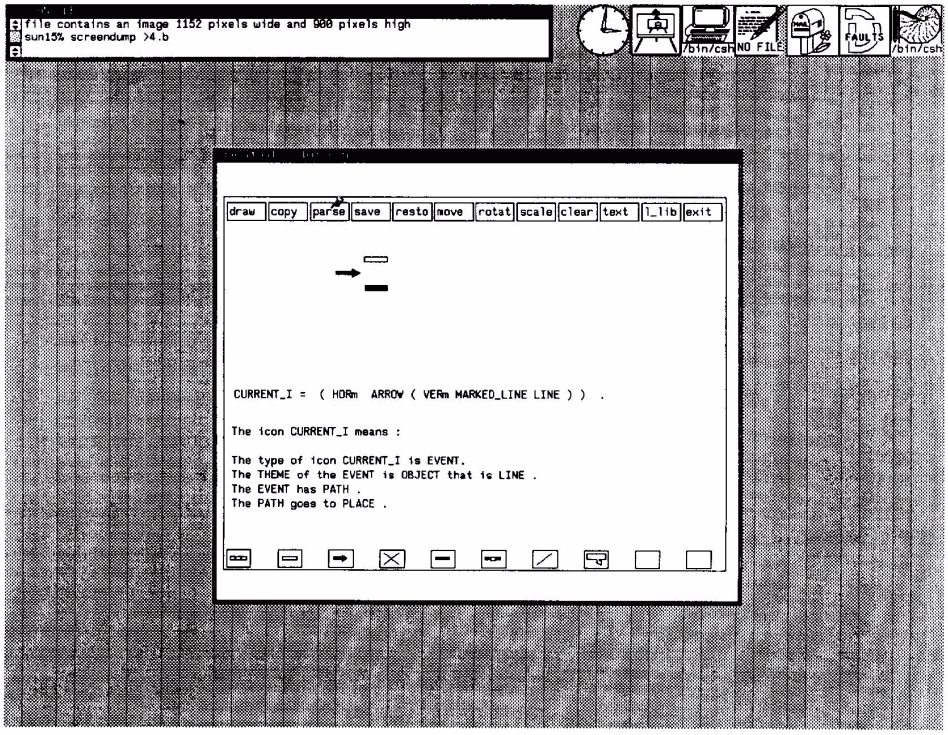
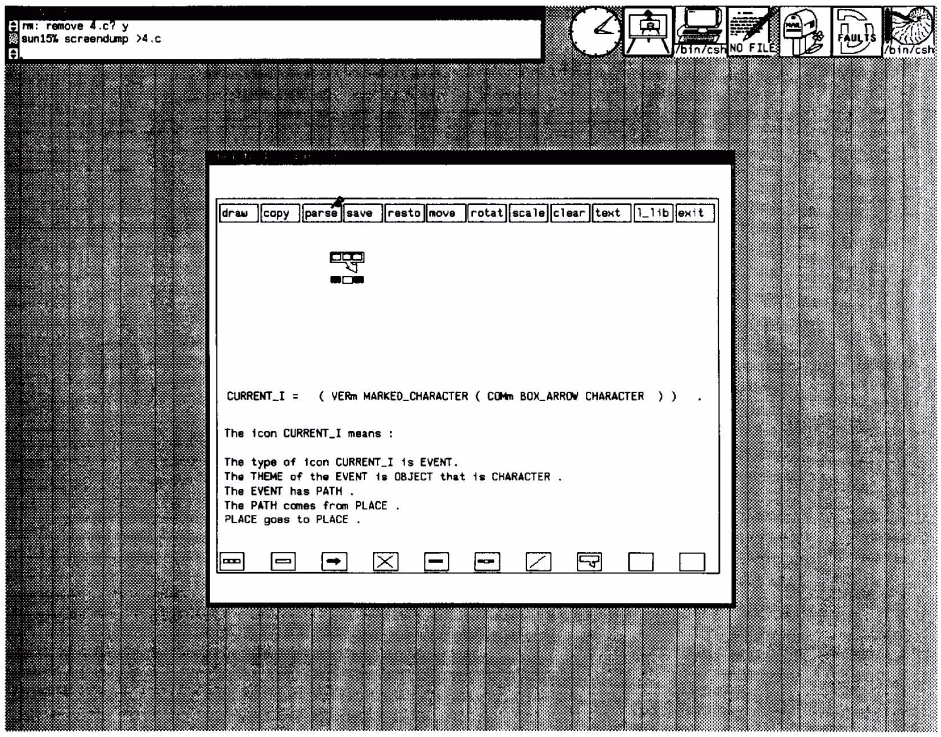
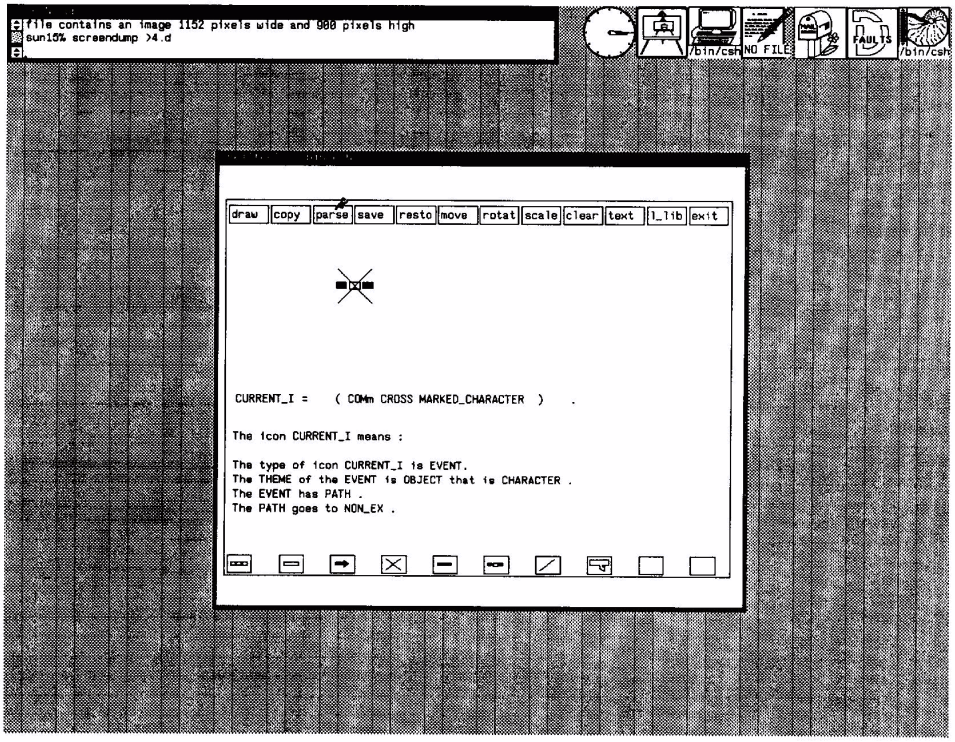


Figure 1.11(b) illustrates the creation of an iconic sentence. This iconic sentence represents the line insertion operation. The iconic sentence is parsed, and the parser output is then displayed. After semantic interpretation, the meaning of this iconic sentence is also displayed.

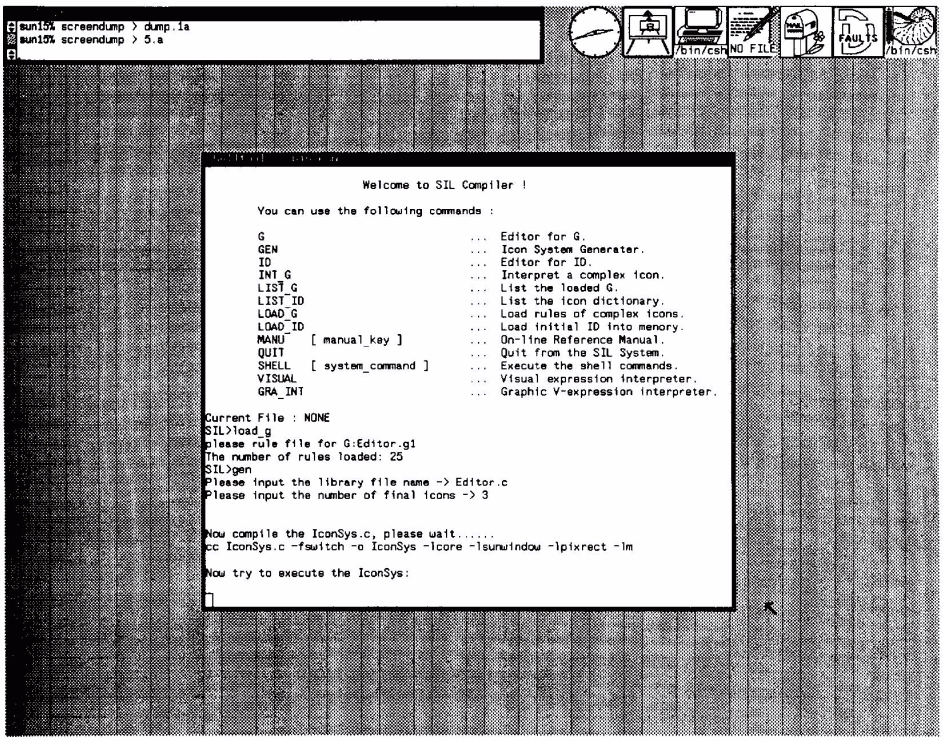


Figures 11(c) and 11(d) are two other examples. The iconic sentence in Figure 1.11(c) represents the string replacement operation, and the iconic sentence in Figure 1.11(d) represents the string deletion operation. The parsed expression and the semantic interpreta tion are also shown.

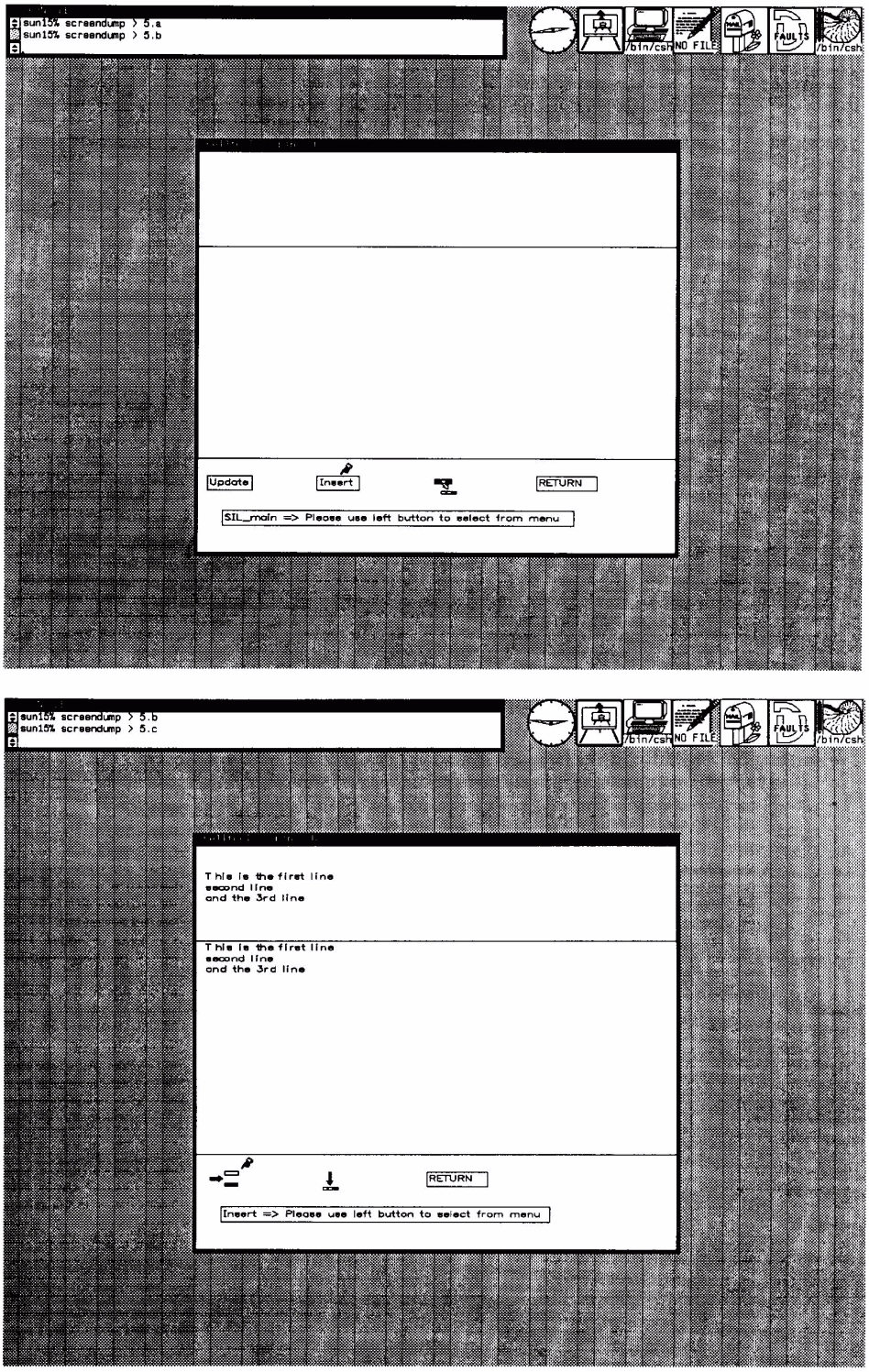




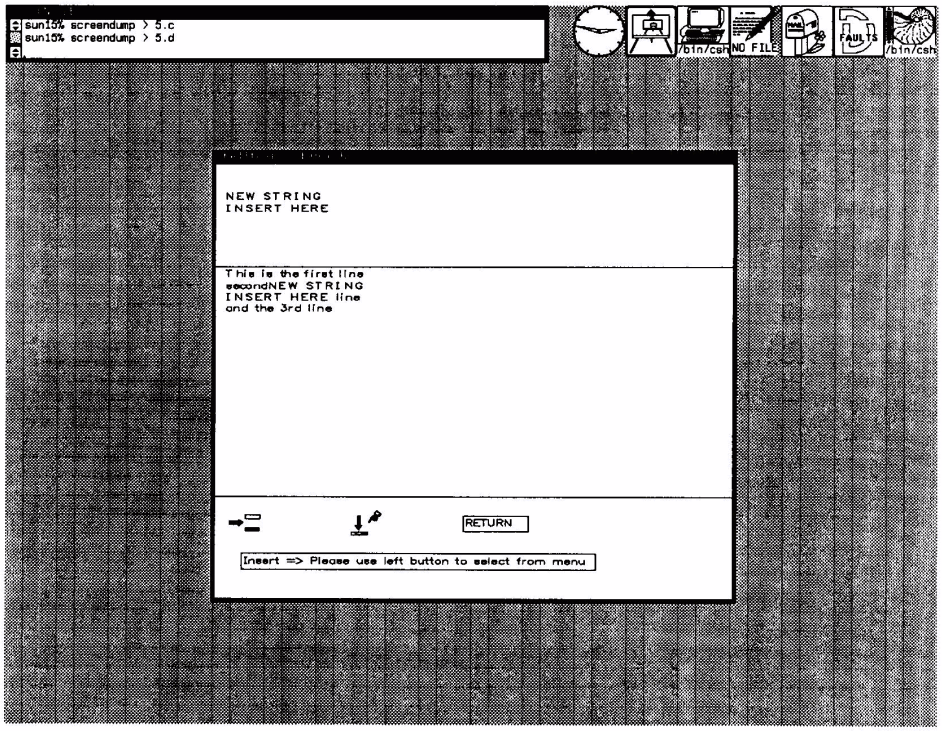
In Figure 1.12(a), the user enters the SIL-ICON compiler and selects the "load_g" function to load the icon grammar G. The user then selects the "gen" function to invoke the icon system generator. The icon system generator requires a library file ("Editor.c") so that it can create the customized user interface program ("IconSys.c"). This source code is compiled, and then the user can execute the user interface ("IconSys").



In Figure 1.12(b), the screen layout of the customized user interface (a simple screen editor) is shown. The user can select the "insert" menu function, and Figure 1.12(c) shows the screen layout of the functions available under the "insert" menu. The user can select the "Insert_line" function to enter text.



In Figure 1.12(d), the user selects the "Insert_string" function to insert string.



in Figure 1.13. The visual query shown is identical to the first iconic sentence shown in Figure 1.2.

The first version of the SIL-ICON compiler, SIL.V1, runs on the VAX780 computer and can only generate text-oriented user interface. The second version, SIL.V2, runs on the SUN workstation and utilizes the graphics package SUNCORE and SUNVIEW to generate icon-oriented user interface. Both versions are written in the C programming language. A third version, SIL.V3, incorporating more efficient parser and interpreter, is currently under development at the Visual Computer Laboratory of University of Pittsburgh.

1

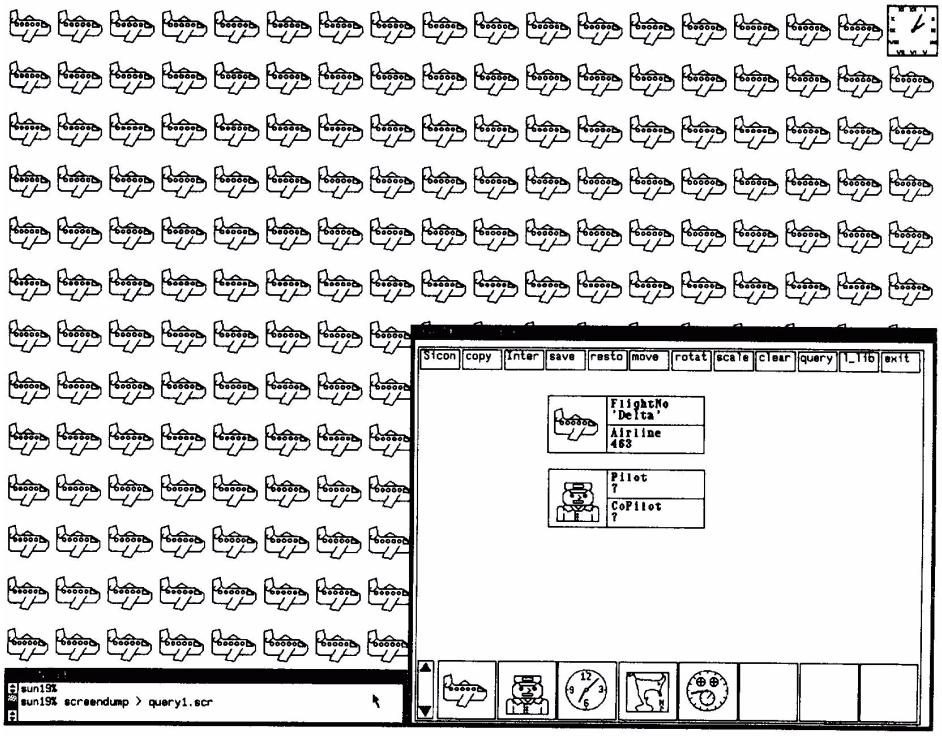


Figure 1.13 Visual query interface.

6. Discussion

Formal icon specification and the icon algebra form the basis of a design methodology for visual languages and icon-oriented systems. The formal icon specification handles the static aspect of an iconic system. The icon algebra handles the dynamic aspect of an iconic system. The iconic indexing technique, together with the concept of icon similarity, allow the indexing and comparison of generalized

icons.

With a powerful icon algebra, we can write programs using these very-high-level operators, to perform visual programming, image processing, image database design, document editing, robotic manipulation, VLSI design, etc.

In application to document editing, the icon (Xm,Xi) represents documents. In (Xm,Xi), Xm is document structure and meaning, and Xi is external document presentation. A generic operator DOCUMENT(op, X, Y) then defines how documents can be combined and its semantic effects. The dematerialization operator DMA(Xi) gives the *symbolic reference* to document Xi. In document editing, the indexing operator can be used to reduce generalized icons "with histories" into some canonical form. In this way, each user may be dealing with a specialized document (including its history of changes), but a common canonical copy can be kept for public reference.

In application to robotic manipulation, the icon (Xm,Xi) may represent a robotic arm. In (Xm,Xi), Xm denotes the virtual arm, and Xi denotes the physical arm. Therefore, a generic robotic operator ROBOT(op, X) can be defined, to manipulate the logical arm Xm and the physical arm Xi. In general, any agent can be thought of as an icon (Xm,Xi), with Xm representing the abstract model, and Xi the physical realization.

We can apply iconic system to VLSI design, where a VLSI icon represents a (design-specification, physical-layout) combination. In other words, in (Xm,Xi), Xm is VLSI design specification, and Xi is physical layout. We can have a generic iconic operator VLSI(op, X, Y) = (VLSIm(op, Xm,Ym), VLSIi(op, Xi,Yi)), where VLSIi(op, Xi,Yi) specifies how the two layouts are combined, and VLSIm(op, Xm,Ym) specifies how the two design specifications are merged.

The concept of generalized icons and icon algebra also provides a unified framework for the theory of pattern recognition. Conceptually, we can compare statistical pattern recognition, syntactical pattern recognition, and clustering analysis as follows:

In statistical pattern recognition, we are given an image ({},pj) and we want to classify the image into class ci, and obtain ({ci},pj). The transformation is:

ENH(({},pj)) = ({ci},pj)

where ENH is an enhancement iconic operator.

In syntactical pattern recognition, we are given an image ({},pj) and we want to construct a formal iconic system G, whose head icon is xo. The transformation is:

SYN(({},pj)) = xo

where SYN is a synthesizing process to construct an iconic system G.

In clustering analysis, we are given images {p1,...,pn}, and we want to cluster them into classes {c1,...,cm}. The transformation is:

```
CLU( ({c1},e),...,({cm},e),({},p1),...,({},pn) )
= { ({ci},pj): 1=<i=<m, 1=<j=<n }
```

As described above, the theory of icons is applicable to the design of icon-oriented user interfaces. A visual language compiler can be constructed, accepting the formal iconic system, the icon operators, and definition of basic icons as input. It can then parse an iconic sentence to determine its syntactic structure and semantic meaning. A realized icon-oriented system can be generated by the visual language compiler, if the initial design is found to be satisfactory.

The theory of icons, or the theory of dual representations of objects and its semantics, therefore can be seen to be a unified methodology for visual language design as well as icon-oriented system design.

APPENDIX 1: THE HEIDELBERG ICON SET

APPENDIX 1.1: ELEMENTARY ICONS (ID)

LINE, rectangle,"a line", OBJECT,,
[OBJECT = LINE].

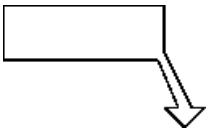
MARKED_LINE, d_rectangle,"a focused line", OBJECT,,
[OBJECT = LINE = MARKED-LINE].

CHARACTER, square,"a character", OBJECT,,
[OBJECT = CHARACTER].

MARKED_CHARACTER, d_square,"a focused character", OBJECT,,
[OBJECT = CHARACTER = MARKED-CHARACTER].

EXCHANGE, diagonal,"object defined place for object exchange", PLACE,,
[PLACE = EXCHANGE_REGION] -> (ON-REGION.[OBJECT:*x]) -> [OBJECT:*y].

ARROW, d_arrow; u_arrow; r_arrow,"location of new objects", EVENT,,
[EVENT = INSERTION] -- (THEME) -> [OBJECT]
(event.GO) -> [PATH = P.1] -> (path.TO) -> [PLACE].



BOX_ARROW, b_arrow,"movement of objects", EVENT,,
[EVENT = MOVEMENT] -- (THEME) -> [OBJECT]
(event.GO) -> [PATH = P.3] -> (path.FROM) -> [PLACE]
(path.TO) -> [PLACE].

CROSS, cross,"deletion of objects", EVENT,,
[EVENT = DELETION] -- (THEME) -> [OBJECT]
(event.GO) -> [PATH = P.2] -> (path.TO) -> [NON-EX].

GAP, gap,"place between two objects", PLACE,,
[PLACE = BETWEEN] -> (place.BEFORE) -> [OBJECT].

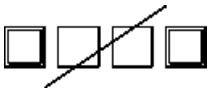
APPENDIX 1.2: INTERMEDIARY ICONS

ON_ROW, fpp(LINE) ^ fpp(GAP) ^ fpp(MARKED_LINE),"place for a line", PLACE,
[PLACE = ROW] -> (ON-ROW.[OBJECT = LINE]) -> [OBJECT = TEXT];

PLACE,,
[PLACE = ROW] -> (BEFORE) -> [OBJECT = LINE].

ON_POINT, fpp(CHARACTER) + fpp(GAP) + fpp(MARKED_CHARACTER),
"place for a character", PLACE,
[PLACE = POINT] -> (ON-POINT.[OBJECT = CHARACTER]) -> [OBJECT = TEXT];
PLACE,,
[PLACE = POINT] -> (BEFORE) -> [OBJECT = CHARACTER].

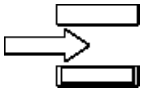
MARKED_STRING, fpp(MARKED_CHARACTER) + fpp(CHARACTER) + fpp(MARKED_CHARACTER);
fpp(MARKED_CHARACTER) + fpp(CHARACTER) + fpp(CHARACTER) + fpp(MARKED_CHARACTER),
"focused string", OBJECT,,
[OBJECT = STRING].



EXCHANGE_REGION,fpp(EXCHANGE) & fpp(MARKED_STRING), "string defining an exchange place",
PLACE,, [PLACE = REGION] -> (ON-REGION.[OBJECT = STRING]) -> [OBJECT = TEXT]

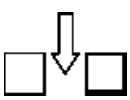
CREATED_STRING, fpp(CHARACTER) + fpp(CHARACTER) + fpp(CHARACTER),
"string to be created", OBJECT,,
[OBJECT = STRING].

APPENDIX 1.3: THE FIVE EDITING ICONS



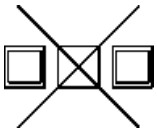
INSERT_LINE, r_arrow + fpp(ON_ROW),"insert a line", EVENT,,

```
[EVENT = INSERT_LINE] -- (THEME) -> [OBJECT = LINE]

                         (event.GO) -> [PATH = P.1.1] -> (path.TO) -> [PLACE = ROW].
```



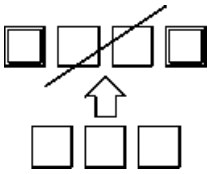
INSERT_STRING, d_arrow ^ fpp(ON_POINT),"insert a string", EVENT,,

```
[EVENT = INSERT_STRING] -- (THEME) -> [OBJECT = STRING]

                           (event.GO) -> [PATH = P.1.2] -> (place.TO) -> [PLACE = POINT].
```



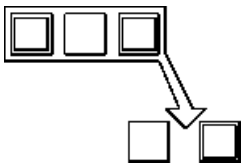
DELETE_STRING, fpp(CROSS) & fpp(MARKED_STRING),"delete a string", EVENT,,

```
[EVENT = DELETE_STRING] -- (THEME) -> [OBJECT = STRING]

                          (event.GO) -> [PATH = P.2] -> (path.TO) -> [NON-EX].
```



REPLACE_STRING, ((fpp(EXCHANGE) & fpp(MARKED_REGION)) ^ u_arrow) ^
fpp(CREATED_STRING),"replace a string", EVENT,,

```
[EVENT = REPLACE_STRING] -- (THEME) -> [OBJECT = STRING]

                           (event.GO) -> [PATH = P.1.3] -> (path.TO) -> [PLACE = REGION].
```



MOVE_STRING, (fpp(MARKED_STRING) & fpp(BOXED_ARROW)) ^
fpp(ON_POINT),"move a string to a new place", EVENT,,

```
[EVENT = MOVE_STRING] -- (THEME) -> [OBJECT = STRING]

                         (event.GO) -> [PATH = P.3.1] -- (path.FROM) -> [PLACE =  REGION]

                                                  (path.TO) -> [PLACE = POINT].
```

APPENDIX 2: FORMAL ICON GRAMMAR G FOR THE HEIDELBERG ICON SET

In the grammar rule for the Heidelberg Icon Set, each right-hand side consists of three components, specifying the logical part, the physical part and the evaluation part. The evaluation part is represented as a string which can be mapped to an implementation dependent object related to the evaluation as discussed in Section 3. The *nil* evaluation part means inheritance. Of course, the elementary icon with nil evaluation part simply means no operation.

APPENDIX 2.1: ELEMENTARY ICON SET

The elementary icon set in ID is predefined by the system manager.

```
 E1. Line ::= ( LINE, rectangle, nil );

 E2. Marked_line ::= ( MARKED_LINE, d_rectangle, "Line_Pointer" );

 E3. Character ::= ( CHARACTER, square, nil );

 E4. Marked_char ::= ( MARKED_CHARACTER, d_square, "Char_Pointer" );
```

```
E5. Exchange ::= ( EXCHANGE, diagonal, "Generic_Exchange" );

E6. Up_arrow ::= ( ARROW, u_arrow, "Generic_Arrow" );

E7. Down_arrow ::= ( ARROW, d_arrow, "Generic_Arrow"  );

E8. Right_arrow ::= ( ARROW, r_arrow, "Generic_Arrow" );

E9. Box_arrow ::= ( BOX_ARROW, b_arrow, "Boxed_Arrow" );

E10. Cross ::= ( CROSS, cross, "Generic_Kill" );

E11. Gap ::= (  GAP, gap, nil).
```

APPENDIX 2.2: INTERMEDIARY ICONS

The intermediary icons are complex icons produced from the elementary icons, but are used only for the definition of more complex icons. Most intermediary icons are predefined by the system manager. They can also be defined or redefined by the end user.

```
I1. On_row ::= ( {VER_m, Line, Gap, Marked_line},
   {VER_i, Line, Gap, Marked_line},
   nil );

I2. On_point ::= ( {HOR_m, Character, Marked_char},
   {HOR_i, Character, Marked_char},
   nil );

I3. Marked_string ::= ({HOR_m, Marked_char, Character, Marked_char},
   {HOR_i, Marked_char, Character, Marked_char},
   nil );

I4. Marked_region ::= ({HOR_m, Marked_char, Character, Character, Marked_char},
   {HOR_i, Marked_char, Character, Character, Marked_char},
   nil );

I5. Created_string ::= ({HOR_m, Character, Character, Character},
   {HOR_i, Character, Character, Character},
   "Create_String" );

I6. Exchange_region ::= ({COM_m, Exchange, Marked_region},
   {COM_i, Exchange, Marked_region},
   nil );

I7. Boxed_string ::= ({COM_m, Box_arrow, Marked_string},
   {COM_i, Box_arrow, Marked_string},
   nil );
```

APPENDIX 2.3: TOP-LEVEL COMPLEX ICONS FOR THE HEIDELBERG EDITOR

The last five rules define the Heidelberg Icon Set.

```
C1. Insert_line ::= ( {HOR_m, Right_arrow, On_row},
    {HOR_i, Right_arrow, On_row},
    "Insert_line" );
C2. Insert_string ::= ({HOR_m, Down_arrow, On_Point},
    {HOR_i, Down_arrow, On_Point},
    "Insert_string" );
C3. Delete_string ::= ({COM_m, Cross, Marked_string},
    {COM_i, Cross, Marked_string},
    "Delete_string" );
C4. Replace_string ::= ({VER_m, Up_arrow, Created_string, Exchange_region},
    {VER_i, Up_arrow, Created_string, Exchange_region},
    "Replace_string" );
C5. Move_string ::= ({VER_m, Boxed_string, On_point},
    {VER_i, Boxed_string, On_point},
    "Move_string" );
```

In the above rules, if we define explicitly the evaluation procedure for each complex icon,
such definition will override the evaluation procedure produced from ID and OD.
If the ID and OD are well defined, and the meaning of complex icons are derivable from ID using operators in OD, then the user can use a
nil evaluation part to obtain the icon interpretation produced by SIL-ICON Compiler.

APPENDIX 3: THE FORMAL GRAMMAR IG OF ICON DICTIONARY ENTRIES

ENTRY-->NAME,SKETCH{;SKETCH},DESC,TYPE,EVAL,CG{;TYPE,EVAL,CG},ATTRIBUTE.
NAME-->STRING
DESC-->[STRING { BLANK STRING}]
TYPE-->STRING
EVAL-->[STRING]
CG-->CONCEPT [RLINK] | RELATION CONLINK
ATTRIBUTE-->[STRING{;STRING}]
STRING-->LETTER STRING | LETTER
LETTER-->A | B | ... | Z | _ | a | b | ... | z
BLANK-->''

In the above, the SKETCH is a pattern string described by the picture grammar PG.

References:

[CHANG70] S. K. Chang, "A Method for the Structural Analysis of Two-Dimensional Mathematical Expressions", *Information Sciences,* Vol. 2, 1970, 253-272.

[CHANG71] S. K. Chang, "Picture Processing Grammar and its Applications", *Information Sciences,* Vol. 3, 1971, 121-148.

[CHANG84] S. K. Chang and S. H. Liu, "Picture indexing and Abstraction Techniques for Pictorial Databases", IEEE Transactions on Pattern Analysis and Machine Intelligence, July 1984, 475-484.

[CHANG85] S. K. Chang, E. Jungert, S. Levialdi, G. Tortora, and T. Ichikawa, "An Image Processing Language with Icon-Assisted Navigation", IEEE Transactions on Software Engineering, August 1985, 811-819.

[CHANG86] S. K. Chang et. al., "Iconic Indexing by 2D Strings", Proceedings of IEEE Workshop on Visual Languages, Dallas, Texas, June 25-27, 1986.

[CHANG87] S. K. Chang, G. Tortora and B. Yu, "Icon Purity - Toward a Formal Theory of Icons", Proceedings of IEEE Workshop on Visual Languages, Linkoping, Sweden, August 1987.

[CHI85] Chi, U.L. "Formal Specification of User Interfaces: A Comparison and Evaluation of four Axiomatic Approaches", IEEE Transactions on Softwareengineering, SE-11, 8. pp.671 - 685.

[CLARISSE85] O. Clarisse and S. K. Chang, "An Icon Manager in Lisp", Proceedings of 1985 IEEE Workshop on Languages for Automation, Mallorca, Spain, June 28-29, 1985, 116-131.

[EDMONDS82] Edmonds, E.A. "T Man-Computer Interface - A Note on Concepts and Design", Int.J.Man-Machine Studies, 16, pp 231 - 236.

[FILE83] G. File, "Interpretation and Reduction of Attribute Grammars", Acta Informatica 19, 115-150, 1983.

[FU74] K. S. Fu, "Syntactic Methods in Pattern Recognition", Academic Press, 1974.

[GREEN85] Green, M. "Design Notations and User Interface Management Systems", in Pfaff, G.E., User Interface Management Systems, Springer: Berlin - Heidelberg - New York - Tokyo.

[GREEN87] T.R.G. Green, F. Schiele & S.J. Payne, "Formalizable Models of User Knowledge in Human - Computer Interaction", to appear in: Green, Hoc et al. (Eds.), Theory and Outcomes in Human-Computer Interaction, Academic Press, London 1987.

[GUEST82] Guest, S.P. "The Use of Software Tools for Dialogue Design', Int.J.Man-Machine Studies, 16, pp 263 - 285.

[HOPPE86] H. U. Hoppe, M. Tauber, J. E. Ziegler, "A Survey of Models and Formal Description Methods in Human-Computer Interface with Example Applications", ESPRIT Project 385 HUFIT, Report B.3.2a, June 1986.

[JACKENDOFF83], R. Jackendoff, "Semantics and Cognition", MIT Press, Cambridge, Mass., 1983.

[JACOBS83] Jacobs, R.J.K. "Using Formal Specifications in the Design of a Human-Computer Interface", Comm.ACM, 26, 4, pp 259 - 264.

[KAMIMURA83] T. Kamimura, "Tree Automata and Attribute Grammars", Information and Control, 57, 1-20, 1983.

[KIERAS83] Kieras, D.E. & Polson, P. "A Generalized Network Representation of Interactive Systems", Proc. CHI'83, Boston, December 1983.

[KIERAS85] Kieras, D.E. & Polson, P. "An Approach to the Formal Analysis of User Complexity", Int.J.Man-Machine Studies, 22, pp 365 - 394.

[KEPPEL&ROHR84] E. Keppel & G. Rohr, "Prototyping - A Method to Explore Human Factors Aspects in Application Software". In: H.W.Hendrik & W.Brown (Eds.): Human Factors in Organizational Design and Management. North Holland, Amsterdam - New York.

[KORFHAGE86] R. R. Korfhage and M. A. Korfhage, "Criteria for Iconic Languages", in Visual Languages, edited by S. K. Chang et. al., Plenum Pub. Co. 1986.

[LODDING82] K. N. Lodding, "Iconics - A Visual Man-Machine Interface", Proceedings of National Computer Graphics Association Conference, Anaheim, California, 1982, Vol. 1, 221-233.

[MORAN81] Moran, T.P. "The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems", Int.J.Man-Machine Studies, 15, pp 3 - 50.

[NORMAN83] Norman, D. A. "Some Observations in Mental Models", In: Gentner et al. (eds.), Mental Models. Erlbaum: Hillsdale, N.J.

[OBER83] Oberquelle, H., Kupka, I. & Maass, S. "A View of Human-Machine Communication and Co-operation", Int.J.Man-Machine Studies, 19, 4, pp 309 - 333.

[PAYNE84] Payne, S. J. "Task Action Grammars", Proc. INTERACT, IFIP, London, September 1984, pp 139 - 144.

[PFAFF85] Pfaff, G. E. "User Interface Management Systems", Springer: Berlin - Heidelberg - New York - Tokyo.

[REISNER84] Reisner, P. H. "Formal Grammars as a Tool for Analyzing Ease of Use: Some Fundamental Concepts", In: Thomas et al. (eds.), Human Factors in Computer Systems. Ablex: Norwood.

[ROHR84] G. Rohr, "Understanding Visual Symbols", Proceedings of IEEE Workshop on Visual Languages, Hiroshima, Japan, December 9-11, 1984.

[ROHR&KEPPEL84] G.Rohr & E.Keppel, "Iconic Interfaces: Where to Use and How to Construct". In: Hendrick, H.W. & Brown, O.jr.

(Eds.). Human Factors in Organizational Design and Management. North Holland, Amsterdam - New York, 1984.

[ROHR86] G. Rohr, "Using Visual Concepts", in: Chang, Ichikawa & Ligomenides (Eds.), Visual Languages, Plenum Press, New York 1986.

[SHNEID86] Ben Shneiderman, *Designing the User Interface,* Addison-Wesley 1986.

[SOWA84] J. F. Sowa, "Conceptual Structures", Addision-Wesley Publ., Reading, Mass, 1984.

[TANIMOTO76] S. L. Tanimoto, "An Iconic/Symbolic Data Structuring Scheme," Pattern Recognition and Artificial Intelligence, Academic Press, 1976, 452-471.

[TANIMOTO86] S. Tanimoto and Marcia S. Runyan, "PLAY: An Iconic Programming System for Children", in Visual Languages, edited by S. K. Chang et. al., Plenum Pub. Co. 1986.

[TAUBER86] M. J. Tauber, "Top-Down Design of Human-Computer Interfaces", in: Chang, Ichikawa & Ligomenides (Eds.), Visual Languages, Plenum Press, New York 1986.

[TAUBER87] M.J. Tauber, "On Mental Models and the User Interface", to appear in: Green, Hoc, Murray & Veer (Eds.): Theory and Outcomes in Human - Computer Interaction, Academic Press, London 1987.

[WASSER85] Wassermann, A.I. "Extending State Transition Diagrams for the Specification of Human-Computer Interaction', IEEE Trans.Software Eng., SE-11, pp 699 - 713.

[WEBSTER83] Webster's New Twentieth Century Dictionary, Unabridged, 1983.

Exercises

(1) Identify the pure icons, impure icons, image icons, label icons, elementary icons and complex icons of the following iconic system:

$x_0 ::= (\{x_1,x_2,x_3\},p_0)$
$x_1 ::= (\{c_1\},p_1)$
$x_2 ::= (\{c_2\},p_1)$
$x_3 ::= (\{x_4,x_5\},p_2)$
$x_4 ::= (\{\},p_3)$
$x_5 ::= (\{c_3\},e)$

(2) Describe how the iconic operators operate on the icons of the formal iconic system G2, illustrated by Figure 1.7 in Section 7, to realize a menu-driven user interface.

(3) Describe the tree menu system of Figure 1.14 as a formal iconic system G3.

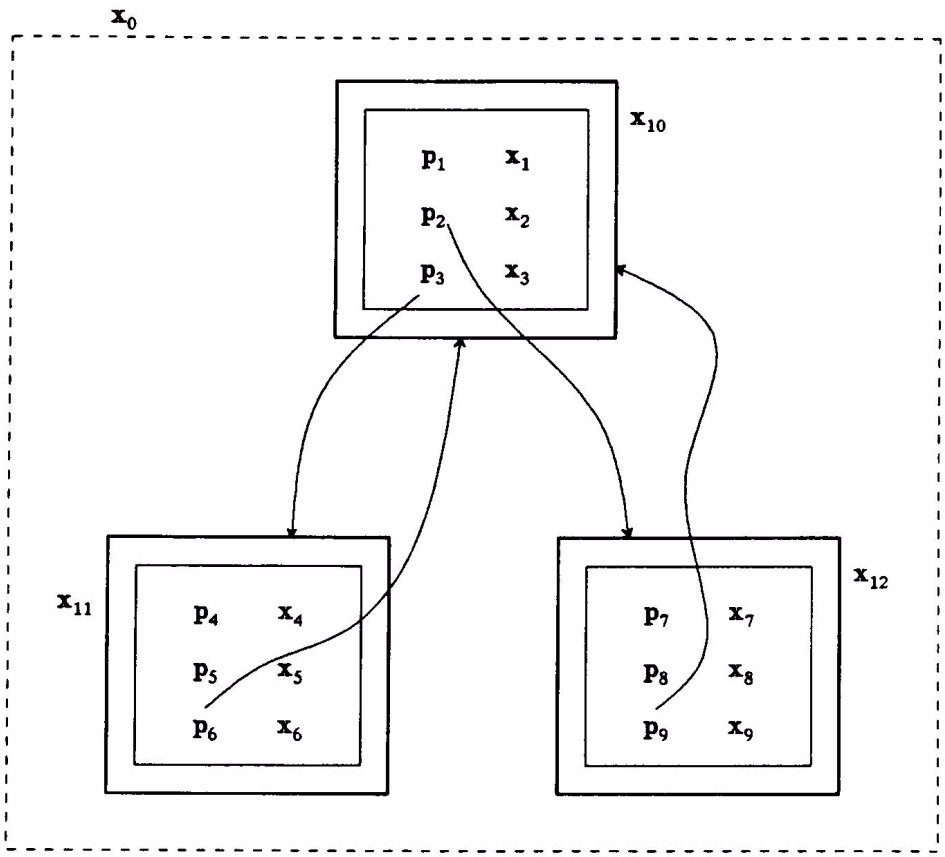


Figure 1.14. A tree menu system.

(4) Perform the syntactic analysis of the complex icon, *delete string,* in the Heidelberg Icon Set (see Figure 1.10), and describe how the parsing tree is obtained.

(5) Perform the semantic analysis of the parsed iconic sentence, *delete string,* and describe step by step the synthesis of the conceptual graph interpretation.

(6) A teleshopping system utilizes home TV monitor to display merchadise, and the shopper can point to the merchandise if further information is needed, or similar types of goods are to be displayed, or shop locations carrying such itmes are desired. (a) Use your imagination to design a set of icons, including elementary icons, operator icons, composite icons, process icons, etc., for this teleshopping system. You should have at least one icon for each type mentioned above. (b) Give a formal description of the above iconic system, using the formal grammar approach. (c) Describe typical iconic sentences in this visual language, and how they are interpreted.
