Chapter 1 Icons and Iconic Languages

Table of Contents

The term "visual language" means different things to different people. To some, it means the objects handled by the language are visual. To others, it means the language itself is visual. To the first group of people, "visual language" means "language for processing visual information", or "visual information processing language". To the second group of people, "visual language" means "language for programming with visual expression", or "visual programming language".

In visual information processing languages, the objects to be dealt with usually have an inherent visual representation. They are images or pictorial objects which are then associated with certain logical interpretation. On the other hand, the languages themselves may not have a visual representation. These languages are usually based upon traditional "linear" languages, enhanced by library subroutines or software packages to deal with visual objects. Application domains of visual information processing languages include image processing, computer vision, robotics, image database management, office automation and image communications.

In visual programming languages, the objects to be dealt with usually do not have an inherent visual representation. They include traditional data types such as arrays, stacks, queues, and application-oriented data types such as forms, documents, databases, etc. To achieve a user-friendly man-machine interface, we would like to present these objects visually. For the same reason, the languages themselves also should be presented visually. In other words, both programming constructs and rules to combine these programming constructs should be visually presented. Application domains of visual programming languages include computer graphics, user interface design, database interface, form management, and computer aided design.

The above two types of visual languages do not exhaust all the possibilities. The objects to be dealt with by a visual language can be inherently visual, or inherently nonvisual but with imposed visual representation. The programming language constructs can be visual or linear. Therefore, there are many different types of visual languages. A unifying concept for these visual languages, is that they all deal with different aspects of *generalized icons.* Generalized icons consist of object icons and process icons. An *object icon* is a dual representation of an object, written as (Xm,Xi), with a logical part Xm (the meaning), and a physical part Xi (the image). In visual programming languages, we are dealing with objects with logical meaning, but no visual image. The objects are then assigned a visual representation, so that it can be visualized. In visual information processing languages, we are dealing with objects with visual image, but the logical meaning must be assigned.

The objects handled by a visual language can thus be considered as object icons or icons with a logical part and a physical part representing an object. Similarly, the programming language constructs in a visual language can be considered as *process icons* or icons with a logical part and a physical part representing a computation process. The distinction between an object icon and a process icon depends both upon context and interpretation. For example, the road sign of a diagonal line inside a circle can be interpreted as a "stop-sign" by a computer vision system. It is an object icon under this latter interpretation. On the other hand, it could also be interpreted as a "halt" command by a mobile robot. It is a process icon (or action icon) under this interpretation. The concept of *generalized icon* encompasses both object icons and process icons (or action icons). We can then study the syntax and semantics of visual languages for both isolated icon and a spatial arrangement of icons.

1. Generalized Icons

The concept of generalized icons, including object icons and process icons, leads to a general approach for designing visual languages. First we ask the question: how can we represent visual objects logically, and conversely, how can we represent logical objects visually? This consideration leads to the concept of object icons. Then we ask the question: how can we represent programming constructs visually and specify algorithms in a visual language? This consideration leads to the concept of process icons (or action icons).

Before we proceed further, let us define what we mean by an icon. The dictionary defines an icon to be "an image; figure; representation; picture" [WEBSTER83]. Icon communication concerns the use of images to convey ideas or actions (commands) in a non-verbal manner. Lodding gives a taxonomy of icons, providing a classification by their design or their function [LODDING82]. By his taxonomy, there are three types of icons distinguished by their design and function:

```
Design   Function
-----    -----
representational  picture
abstract   symbol
arbitrary   sign
```

An icon image is chosen to relate to the idea or action either by resemblance (picture), or by analogy (symbol), or by being selected from a previously defined and learned group of arbitrarily designed images (sign). To assure the correct interpretation of an icon image, there are three requirements:

```
. the right image,
. the right caption,
. the right context.
```

Iconic languages have their problems and drawbacks. As pointed out by Lodding, some icons are inherently ambiguous, some can only be interpreted within a certain context. As pointed out by Korfhage [KORFHAGE86], since there is no commonly accepted universal set of icons, icons may evolve in time. Therefore, the design process of icons must be well thought out. Lodding suggests the design process of icons be divided into three distinct steps or phases: a) choosing the representation, b) rendering the design, and c) testing the resulting icon.

As mentioned above, visual languages can be designed based upon the concept of *generalized icons,* which are dual representations of objects consisting of a logical part and a physical part. Generalized icons can be further classified into *object icons* and *process icons*. The main concepts concerning generalized icons are defined below.

An *iconic system* is a structured set of related icons. A complex icon can be composed from other icons in the iconic system, and therefore express a more complex visual concept. An *iconic sentence* (called visual sentence by Lakin, iconic sentence or action sentence by Tanimoto [TANIMOTO86], and iconic statement by Korfhage) is a spatial arrangement of icons from an iconic system. A *visual language* is a set of iconic sentences constructed with given syntax and semantics. *Syntactic analysis of visual language* (spatial parsing) is the analysis of the spatial arrangement of icons (i.e. an iconic sentence) to determine the underlying syntactic structure. Finally, *semantic analysis of visual language* (spatial interpretation) is the interpretation of an iconic sentence to determine its underlying meaning.

2. Examples of Iconic Systems

Two examples of iconic systems will now be presented. The first example illustrates a visaul database query language. The database contains information on airline flights - flight number, airline, pilot's name, copilot's name, departure time, arrival time, current position of aircraft, current speed of aircraft, etc. The query icons are illustrated in Figure 1.1. Each icon has an image, which is the physical part of the icon. Each icon also has a name and two additional attributes, constituting the logical part of the icon. These icons form an iconic system. They are stored in an icon dictionary.

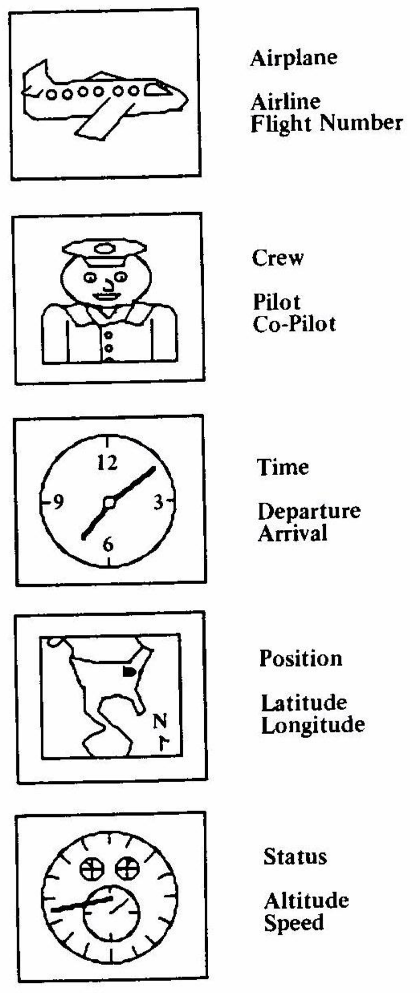

Figure 1.1 Icons for a visual database query language.

To specify a visual query, the user can select some of these query icons and place them in a certain spatial arrangement. The vertical combination of icons denotes logical conjunction, and horizontal combination denotes logical disjunction. By filling in an actual value, an asterisk or a question mark, the user can create an icon instance to indicate a specific condition, a "don't-care" condition, or a retrieval target. Four sample iconic sentences (or visual sentences) from this iconic system are illustrated in Figure 1.2, each denoting a query. For example, the first visual sentence corresponds to the query: "Who are the Crew on Delta 463?" Translated into SQL, the equivalent query is:

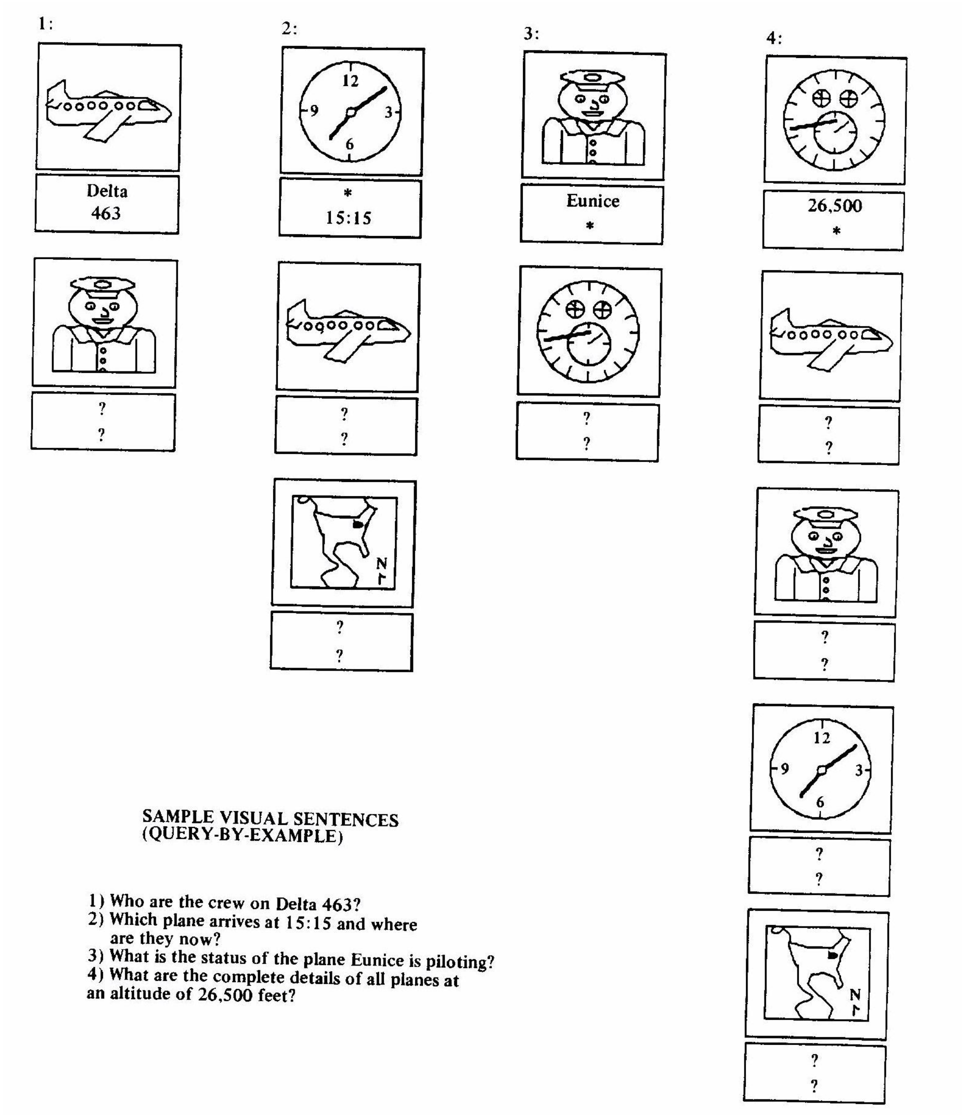
1:
Delta
463
?
?
2:
12
9
3
6
*
15:15
?
?
?
?
3:
Eunice
*
?
?
4:
26,500
*
?
?
?
?
12
9
3
6
?
?
?
?
N
SAMPLE VISUAL SENTENCES
(QUERY-BY-EXAMPLE)
1) Who are the crew on Delta 463?
2) Which plane arrives at 15:15 and where are they now?
3) What is the status of the plane Eunice is piloting?
4) What are the complete details of all planes at an altitude of 26,500 feet?

Figure 1.2 Sample visual queries.

```
SELECT Pilot,CoPilot FROM PlanArr
WHERE Airline = 'Delta' AND FlightNo = 463;
```

This example illustrates how iconic sentences can be constructed using iconic operators, which denote either logical conjunction (vertical combination of icons) or logical disjunction (horizontal combination of icons). In Section 1.4, such iconic operators will be defined formally. It should also be noted that a query icon can become either an object icon (when its attributes are specified) or a process icon (when its attributes are given question marks), depending upon the actual query instance.

The second example illustrates the construction of an iconic system for a pictorial database. Figure 1.3 shows an actual SEASAT picture. By applying picture processing and pattern recognition algorithms, certain shiplike objects in this picture can be reduced to object icons. Some more complex objects, such as the shorearea, can be reduced to complex icons. The picture now becomes a picture consisting in part of the real picture, in part of object icons. These icons are related hierarchically to constitute a tree structure.

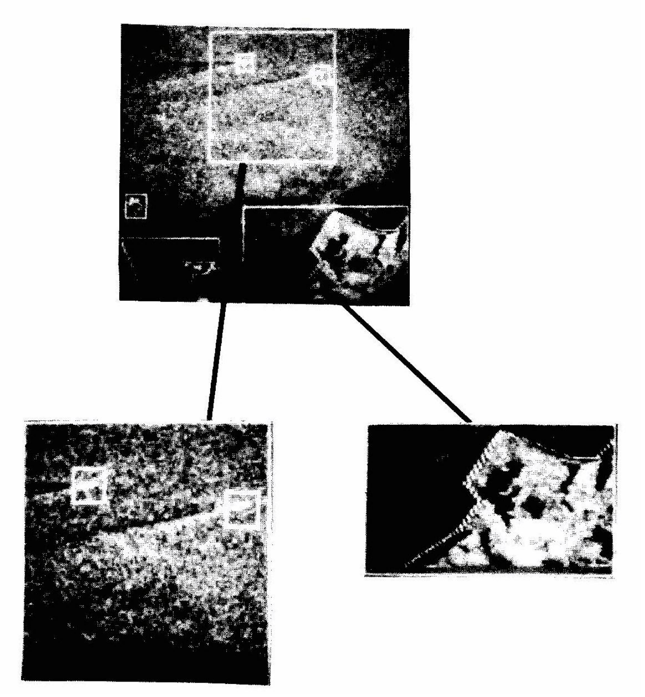
Figure 1.3 Iconic system of a pictorial database.

This example shows that by constructing complex icons, we can create a pictorial database. The resulting hierarchical structure also enables the user to navigate in the pictorial database using zooming techniques.

3. Formal Specification of Iconic Systems

We now present a formal specification of iconic systems using generalized icons. A *generalized icon* is a dual representation of an object, written as (Xm,Xi), with a logical part Xm (the meaning), and a physical part Xi (the image). A formal *iconic system* is a quintuple: G (VL, VP, S, xo, R) where

VL is a set of logical objects

VP is a set of physical objects

S is a finite, nonempty set of icon names

xo is an element in S, denoting the head icon name

R is a mapping from S into $2^{VL \cup S} \times$ VP, denoting icon rules

The icon rules R specify icons as the dual representation by a set of logical objects and a physical object. Several examples will be presented below.

Example 1: For the image database shown in Figure 1.3, the formal iconic system is G1 ({c1,c2}, {po,p1,p2,p3,p4}, {xo,x1,x2,x3,x4}, xo, R1), where the icon rules R1 are:

```
r1: xo ::= ({x1,x2}, po)
r2: x1 ::= ({x3,x4}, p1)
r3: x2 ::= (  { }  , p2)
r4: x3 ::= (  {c1} , p3)
r5: x4 ::= (  {c2} , p4)
```

In the above, rule r1 specifies an icon xo with the logical part {x1,x2} and the physical part po. Both x1 and x2 are icon names in S. Therefore, rule r1 says x1 and x2 are subicons of the icon xo. Rule r2 is similar to rule r1. Rule r3 specifies an icon x2, whose logical part is the null object, and physical part is p2. In other words, x2 is really a "pure image". Rule r4 specifies an icon x4, whose logical part is {c1} and physical part is p3. The object c1 is an element of VL, and is used as a "label" for the physical image p3.

The iconic system can be seen to be a special type of picture grammar [FU74]. The icon grammar rules can also be expressed as commands in an iconic language (see [CHANG85] for a proposed language IPL). We give the following examples to illustrate elements of an interactive iconic language. In what follows, a command line is indicated by a colon. The system response, if any, follows the command line.

```
: X = ICON( Xm, Xi) create an icon X with logical
    part Xm and physical part Xi


: X   display contents of icon X
  ( Xm, Xi )


: VL(X)   display logical part of X
  Xm


: VP(X)   display physical part of X
  Xi


: MAT(Xm)  materialization of logical part Xm
  Xi    as physical part Xi


: DMA(Xi)  dematerialization of physical part Xi
  Xm    as logical part Xm
```

The above iconic language can be implemented using icons in a direct manipulation interface [SHNEID86]. The materialization operator MAT and dematerialization operation DMA will be discussed in the following section. The commands to create the five icons of Example 1 are:

```
x4 = ICON(   {c2} , p4)
x3 = ICON(   {c1} , p3)
x2 = ICON(   { }  , p2)
x1 = ICON( {x3,x4}, p1)
xo = ICON( {x1,x2}, po)
```

Example 2: Suppose we have a book organized as chapters containing sections, as illustrated in Figure 1.4.

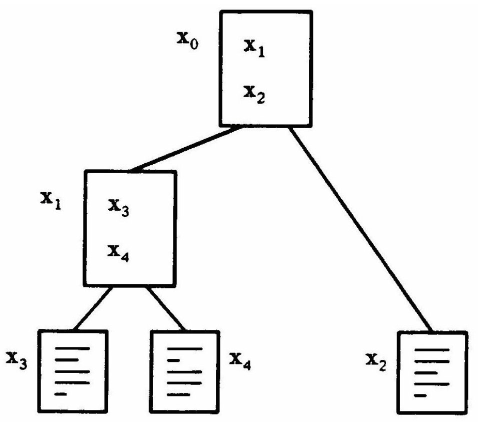

Figure 1.4. Iconic system of a book.

The formal iconic system is G2 ({COM}, {e,p2,p3,p4}, {xo,x1,x2,x3,x4}, xo, R2), where the icon rules R2 are:

```
r1: xo ::= ({COM,x1,x2}, e)
r2: x1 ::= ({COM,x3,x4}, e)
r3: x2 ::= (  { }  ,p2)
r4: x3 ::= (  { }  ,p3)
r5: x4 ::= (  { }  ,p4)
```

The symbol "e" denotes an "empty picture". Therefore, rule r1 specifies an icon xo with subicons x1, x2 and no physical image. The logical part of icon xo is {COM, x1, x2}, where COM is in VL, and x1, x2 are in S. The symbol "COM" is an *iconic operator* which operates on the subicons x1 and x2 to create a new icon. The inclusion of operators as logical objects gives greater flexibility and notational convenience in the specification of the logical part of an icon. The location attributes of the subicons x1 and x2 will determine the order in applying the iconic operator COM.

The command VL(xo) will display the logical part of icon xo, and the system will respond by evaluating and displaying

The command VL(xo) will display the logical part of icon xo, and the system will respond by evaluating and displaying COMm({x1,x2}). The iconic operator COM has two parts: COMm for the logical operator, and COMi for the physical operator (see the following section). In this example, the COMm operator is applied to the icon set {x1,x2} to yield {x1, x2}. Therefore, VL(xo) is equivalent to listing the content of "directory" xo. On the other hand, the command VP(xo) will generate a blank picture, because the physical part of xo is "e".

The command VP(x2) will display the physical part of icon x2, and the system will respond by displaying p2, the content of "file" x2. On the other hand, the command VL(x2) will generate no output, because x2 does not have a logical part.

From these examples, it can be seen that an icon can be a pure physical picture ({}, PICTURE), a pure logical label ({LABEL},e), a complex icon constructed from subicons ({OP, x1,...,xn}, PICTURE), or a complex icon related to subicons with unspecified method of construction ({x1,...,xn}, PICTURE). Therefore, an icon can be one of the following types:

elementary icon:

if $x_m \cap S$ is empty. In other words, x_m is subset of VL, so that x is of the form ({labels}, image). The labels could denote names of objects, procedures (including user-defined procedures, system function calls, etc.), or operators, so that the elementary icon can be an object icon, a process icon, or an operator icon.

There are special elementary icons. An *image icon* is one where x_m is empty, so x is of the form ({ }, image). A *label icon* is one where the physical part is null, so x is of the form ({labels}, e). Finally, a null icon is of the form ({ }, e).

complex icon:

if $x_m \cap S$ is not empty. A complex icon points to other icons and defines icon relations. We can further distinguish the following types:

composite icon: if $x_m \cap VL$ is not empty. The icon x is of the form ({OP,$y_1,\cdots,y_n$}, image), where y_i are subicons or logical objects, and "OP" is an *iconic operator* which operates on the subicons $y_1,\cdots,y_n$ to create a new icon. The inclusion of iconic operators as logical objects gives greater flexibility and notational convenience in the specification of the logical part of a composite icon. The location attributes of the subicons will determine the order in applying the iconic operator.

structural icon: if $x_m \cap VL$ is empty. The icon x is of the form ({$y_1,\cdots,y_n$}, image). In other words, x is related to $y_1,\cdots,y_n$, but the mechanism for composing x from $y_1,\cdots,y_n$ is unspecified.

Given an icon system G, we can classify all icons as being either *elementary, composite* or *structural.* In the above, we described the formal specification of an iconic system, where complex icons can be defined in terms of elementary icons. For complex icons, we need to investigate how to construct an icon with a physical part (the image) and a logical part (the meaning) from subicons by applying iconic operators.

4. Iconic Operators

Iconic operators operate on generalized icons, and change either the logical part (the meaning) or the physical part (the image) of an icon, or both. The concept of duality is essential: *iconic operators operate on the dual representations of icons.* In other words, the iconic operator operates simultaneously on logical and physical parts of an icon. An essential characteristic of the icon is that *the logical part and the physical part are mutually dependent.* Therefore, as the image of an icon is changed, so will be its meaning, and vice versa.

An iconic operator OP has two parts: OPm for the logical operator, and OPi for the physical operator. We write:

OP = (OPm, OPi)

A binary iconic operator OP has two arguments, X and Y:

OP(X,Y) = (OPm(X,Y), OPi(X,Y))

When there is no mutual dependency, i.e., OPm does not depend on Xi or Yi, and OPi does not depend on Xm or Ym, we can write

OP(X,Y) = (OPm(Xm,Ym), OPi(Xi,Yi))

A unary iconic operator OP has only one argument X:

OP(X) = (OPm(X), OPi(X))

Again, when there is no mutual dependency, we can write

OP(X) = (OPm(Xm), OPi(Xi))

Let WL denote $2^{VL \cup S}$. The *materialization operator* MAT is a mapping from WL to 2^{VP}, and the *dematerialization operator* DMA is a mapping from VP to 2^{WL}. A *pure icon* is an icon X = (Xm, Xi), where {Xm} = DMA(Xi) and {Xi} = MAT(Xm).

For pure icons only, the logical part can be completedly recovered from the physical part, and vice versa. This is possible, only when MAT(Xm) and DMA(Xi) are both singletons. In general, MAT(Xm) may yield a set of icon images. For example, MAT("Mona-Lisa") may be the original drawing of Mona-Lisa, or a sketch of Mona-Lisa. DMA(Xi) may also yield a set of meanings. Such impure icons may cause problems if used for person-machine communication. The iconic operators introduced in this section may decrease the purity of icons. In Section 8, we will give purity-preserving conditions for iconic operators.

A *physical iconic operator* operates only on the physical part of an icon, i.e.,

OP(X,Y) = (DMA(OPi(Xi,Yi)), OPi(Xi,Yi))

where X and Y are pure icons.

A *logical iconic operator* operates only on the logical part of an icon, i.e.,

OP(X,Y) = (OPm(Xm,Ym), MAT(OPm(Xm,Ym))

where X and Y are pure icons.

The usual image processing operations can be regarded as physical iconic operators. Similarly, the usual logical operations on objects in a knowledge base can be regarded as logical iconic operators. We now investigate those generic operators which may affect both the meaning and the image of an icon. In what follows, we will use ({Xm},Xi) and (Xm,Xi) interchangeably.

```
(1) Combination COM:


COM((Am,Ai), (Bm,Bi))
= (COMm(Am,Bm), COMi(Ai,Bi))
= (CONCEPT-MERGE(Am,Bm), SUPERPOSE(Ai,Bi))
```

Explanation: The images Ai and Bi are combined by superposition. At the same time, the conceptual merge of the meanings Am and Am becomes the meaning of the resultant new icon. As an example, COM(\h'0.2i', \h'0.2i') = COM((do-not,\h'0.2i'), (road,\h'0.2i')) = (CONCEPT-MERGE(do-not,road), COMBINE(\h'0.2i',\h'0.2i')) = (no-entry,\h'0.2i').

```
(2) Vertical Combination VER:


VER((Am,Ai), (Bm,Bi))
= (VERm(Am,Bm), VERi(Ai,Bi))
= (CONCEPT-MERGE(Am,Bm), VER-COMBINE(Ai,Bi))
```

Explanation: The images Ai and Bi are combined vertically. At the same time, the conceptual merge of the meanings Am and Bm becomes the meaning of the resultant new icon. As an example, the queries shown in Figure 1.2 are vertical combinations of query icons, and the vertical combination denotes conjunction of logical expressions.

```
(3) Horizontal Combination HOR:


HOR((Am,Ai), (Bm,Bi))
= (HORm(Am,Bm), VERi(Ai,Bi))
= (CONCEPT-MERGE(Am,Bm), HOR-COMBINE(Ai,Bi))
```

Explanation: The images Ai and Bi are combined horizontally. At the same time, the conceptual merge of the meanings Am and Bm becomes the meaning of the resultant new icon. As an example, the horizontal combination of "character" icons in the Heidelberg Text Editor Icon Set (see Appendix 1) denotes a "string" icon.

```
(4) Contextual Interpretation CON:


CON((Am,Ai), (Bm,Bi))
= (CONm(Am,Ai,Bm,Bi), CONi(Ai,Bi))
= (CONTEXT(Am,Ai,Bm,Bi), Ai)
```

Explanation: Contextual interpretation is achieved, by considering icon A in the context of icon B. The two icons A and B occur together. The new meaning depends upon both A and B. In the new icon, the image remains to be the image of Ai. Contextual interpretation can obviously incorporate a lot of additional attributes to enhance an icon. Contextual interpretation is most useful in interpreting time-varying iconic sentences, such as video game snapshots.

```
(5) Indexing IDX:


IDX(Am,Ai)
= (IDXm(Am), IDXi(Ai))
= (CONCEPT-REDUC(Am), IMAGE-REDUC(Ai))
```

Explanation: In iconic indexing, the conceptual meaning as well as the image of an icon is simplified, so that a less complicated icon is constructed to serve as an index to the original icon. For the logical part (meaning) of an icon, the simplification is achieved by reducing the conceptual graph, or by tree pruning. For the physical part (image) of an icon, the simplification is achieved by obtaining the sketch, the silhouette, the contour, or the sub-region, of an image. More detailed explanation of iconic indexing will be given in Section 6.

```
(6) Clustering CLU:


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

Explanation: The objects to be clustered, ({},pj), are images. The result of clustering will enhance the meaning of the images. The logical part added, ci, is usually a label. We say image pj is assigned to cluster ci. CLU returns a set of n icons.

```
(7) Cross-Indexing CRO:


CRO( (Am,Ai), (Bm,Bi) )
= ( IDX(A), IDX(B) )
```

Explanation: Cross-indexing is used to relate two icons which are similar according to some criteria They can be physically similar (having similar images), or logically similar (having similar meanings), or a combination of both. CRO returns a pair of icons.

(8) Similarity Operator SIM:

SIM(X,Y) = (SIMm(Xm,Ym), SIMi(Xi,Yi))

Explanation: Similarity operator returns a "true" icon (true-m, true-i), or a "false" icon (false-m, false-i). True/false icons are pure icons. If SIM(X,Y) depends only on SIMi(Xi,Yi), the similarity operator tests the physical similarity of two images. On the other hand, if SIM(X,Y) depends only on SIMm(Xm,Ym), the similarity operator tests the logical similarity of two images. The logical similarity of icons allows for variations of icon images. For example, (stop,\h'0.2i') and (stop,\h'0.2i') are considered similar by logical similarity. (diner,\h'0.2i'), (diner,\h'0.2i'), (diner,\h'0.2i') are also considered similar by logical similarity. If a similarity operator returns (true-m,false-i) or (false-m,true-i), it is considered meaningless.

(9) Existence Operator EXI:

EXI(X,Y) = (EXIm(Xm,Ym), EXIi(Xi,Yi))

Explanation: The existence operator tests for the existence of X in Y, and returns a true/false icon. We can test for the existence of the logical object Xm within Ym, or the existence of the physical object Xi within Yi, or both.

The generic iconic operators discussed above can be used to construct arbitrarily complex icons. These operators are generic, in the sense that their semantics are not completely specified. For specific formal iconic systems, we can use "custom-made" iconic operators to interpret complex icons.

5. Syntactic Analysis of Iconic Sentence

Iconic sentences or complex icons are constructed from elementary icons using iconic operators. The syntactic analysis of an iconic sentence, or spatial parsing, can be based upon the methodology of picture grammars and precedence grammars [CHANG70, CHANG71, FU74]. Picture grammar was originally designed to parse digital pictures on square grid. It is based on the fact that digital pictures consist of picture elements (pixels). This can be demonstrated by following example. Suppose we have an object set: a point (2,3) and a line with the coordinates of two end points (4,3) and (8,3). Then a set of grammar rules can be used to describe them:

```
1) h-dash(4,3,2) := point(4,3,1) & point(5,3,1)
2) h-dash(6,3,2) := point(6,3,1) & point(7,3,1)
3) h-line(4,3,4) := h-dash(4,3,2) & h-line(6,3,2)
4) h-line(4,3,5) := h-line(4,3,4) & point(8,3,1)  and
```

$$5)\ \underset{\delta =< 1}{\text{h-line}}(2,3,1+5+\delta) := \text{point}(2,3,1)\ \&\ \text{h-line}(4,3,5)$$

where δ is a gap tolerance in a line. Similarly we can construct other grammar rules for lines other than horizontal lines, and for any other shapes.

Now we come to the general definition of picture processing grammar:

A *picture-processing grammar* G is a quintuple(S,V,C,g,P), where S is the set of basic symbols, V is the set of vocabulary symbols, C *!supset* V is the set of categorical symbols, g is a function from V $\cup$ S into the set of natural numbers, and P is the set of grammar rules. Each rule is of the from

$$\alpha\ (f(x_1, x_2, \ldots, x_k)) := \beta_1(x_1)\ \&\ \beta_2(x_2)\ \&\ldots\&\ \beta_k(x_k)$$

where $\beta_1, \beta_2, \ldots, \beta_k \in V \cup S,\ \alpha \in V$, the number of parameters of the associate vector $\bar{x}_i$ is equal to $g(\beta_i),\ 1 \le i \le k$, and f is a partially computable function from $I^{\sum_{i=1}^{k} g(\beta_i)}$ into $I^{g(\alpha)}$, whose completion is also computable. It is required that $k \ge 1$.

According to the definition, rule 1) can be rewritten in general form as

h-dash[(4,3,1),(5,3,1)] := point(4,3,1) & point(5,3,1)

A picture processing grammar may have a hierarchical structure as defined below:

A picture processing grammar G=(S,V,C,g,P) is called *hierarchical* if and only if there is a nontrivial partition of the rules P into blocks $R_1, R_2, \ldots, R_n$, n>1, such that if α appears as the left-hand symbol of a rule in R_i, then it will never appear as a right-hand symbol of any rule in R_j, provided that j<i.

The parsing algorithm of the languages generated by hierarchical picture processing grammar is then as follows:

```
1) Partition the set of rules P into hierarchical blocks $R_1, R_2, \ldots, R_n$,
2) i $\leftarrow$ 1,
        3) Apply rules in $R_i$ in any sequence to reduce the picture.
When no more reduction is possible, go to the next step,
        4) If i=n or the picture has been reduced to a categorical symbol,
stop.  Otherwise let $i \leftarrow i+1$ and go to step 3).
```

The picture processing grammar described above was successfully applied to analyze hand-written numericals, where the grammar rules are divided into four groups. The first level rules are used to reduce a picture consisting of points to one consisting of horizontal strips. Imperfect strips can be recognized by adjusting δ to fill gaps. The second level rules are used to reduce the picture to one consisting of vertical lines (V-L) and horizontal lines (H-L). In the third level, L-shaped, U-shaped, and O-shaped figures are described. Finally, in the fourth or last level, the ten numerals are described in terms of their component parts.

Precedence grammar [CHANG 70] is another spatial parsing grammar and can be used for 2-D mathematical expression analysis and printed page format analysis. Precedence grammars are in fact more suitable for the syntactic analysis of iconic sentences, because iconic sentences are constructed using elementary icons and iconic operators. Let us start explaining it by giving a simple example. Suppose we have a mathematical expression a + (b). The structure specification scheme is then defined to describe these mathematical expressions. Assume that some or all primitive components are operators, each of them is uniquely associated with a region, a rectangular area on a 2-D plane, mathematically written as $L(X_c, Y_c, X_{min}, Y_{min}, X_{max}, Y_{max})$ where X_c, Y_c are coordinates of region centroid. A pattern U is a finite collection of primitive components whose regions are disjoint. A frame F of a pattern U is the smallest region containing all primitive components of U.

The structure tree is then constructed by comparing precedence of operators in a pattern and dividing the pattern into one or several subpatterns. Further division are carried out on those sub-patterns. Figure 1.5(b) is the structure tree of expression a + (b) and Figure 1.5(a) is the frame partition corresponding to Figure 1.5(b). The operators "(" and ")" are grouped together because they are regarded as an operator pair. The structure specification scheme is then defined as finite collection of these division rules. Thus we have following formal definition:

A well-formed structure of a pattern U is a tree T whose nodes (or node-name) are operator sequences and regions such that

```
        1) The root of the tree is the frame F of U,
        2) Every primitive component of U appears as a member of some operator
sequence that is a terminal node of T, and the set of all operator sequences
of T is U,
        3) If a region is a terminal node, then it is a non-essential region
with respect to some division rule,
        4) If a region is a non-terminal node, then its successor nodes
constitute a division of it with respect to some division rule,
        5) If operator $w_1$ and region R are successor nodes of some
node and operator $w_2$ is a successor node of R, then $w_1$ dominates
$w_2$ or $w_1$ procedes $w_2$ or they are of equal precedence.
```

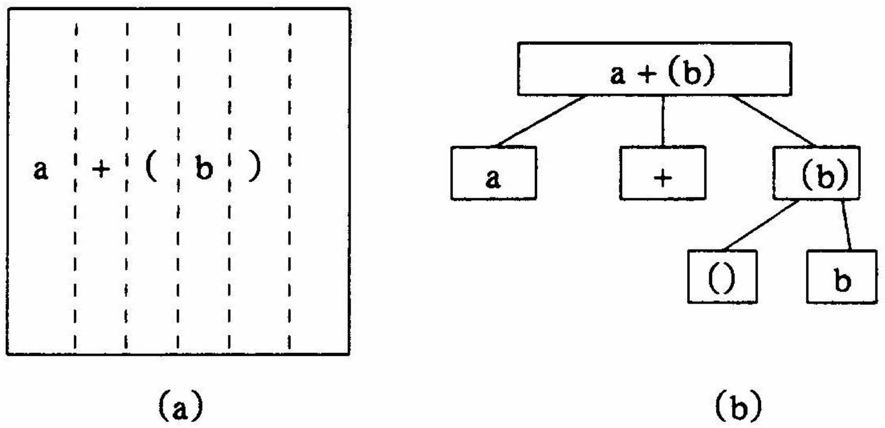

Figure 1.5. (a) A partition of the expression "a + (b)", (b) structure tree of the expression "a + (b)".

According to above definition the structural parsing is carried out by first grouping the primitive components of U into operator sequences, and then building up a structural tree T of U. To build up a structure tree T of U we let F_0 be the frame of U. Let $w_1, w_2, \ldots, w_n$ be all the operator sequences of U. Construct two $n \times n$ precedence matrices M1 and M2 as follows: M1(i,j) is ">" if w_i dominates w_j, "<" if w_j dominates w_i, and blank otherwise. M2(i,j) is ">" if w_i precedes w_j, "<" if w_j precedes w_i, "=" if they are of equal precedence, and blank otherwise. All comparisons are done with respect to F_0. By using these two matrices M1 and M2, we choose w_i to divide F_0 with respect to w_i, such that w_i is in F_0 and w_i is not dominated by any other w_j in F_0 and F_0 can be divided with respect to w_i into essential and non-essential regions that contain all other w's in F_0.

In the example shown in Figure 1.5(b), the operator "+" is not dominated by any other operator. Therefore, it is selected as the first operato be applied tor to divide the frame.

In an iconic sentence, the iconic operators are often "invisible". resultant icon image. Therefore, in analyzing an iconic sentence, such "invisible" operators must be restored, so that the technique of spatial dominance and precedence analysis described above can be applied.

We can first find the minimum enclosing rectangles (MERs) of the elementary icons in an iconic sentence. By comparing the MERs, invisible operators can be restored. For example, the most commonly used iconic operators are COM, HOR and VER. If two MERs coincide to a considerable degree, then the operator COM is added. If one MER is on top of another MER, then the operator VER is added. If one MER is to the left of another MER, then the operator HOR is added. Precedence matrices M1 and M2 are then constructed to order the iconic operators. Care must be taken to discard "double" iconic operators which are created in comparing the MERs, so that the input iconic sentence can be analyzed correctly. If we don't use the precedence analysis technique, a general parsing algorithm must be used to parse the iconic sentence according to G.

6. Semantic Analysis of Iconic Sentence

The result of the syntactic analysis of an iconic sentence is a parsing tree, which can be described by an iconic system, with the head icon representing the entire iconic sentence (or the complex icon to be analyzed). Therefore, in performing the semantic analysis of iconic sentence, we are actually evaluating the meaning of an iconic system corresponding to that iconic sentence.

To evaluate an iconic system G, we can apply a generic iconic operator INT recursively to construct an interpretation of the formal iconic system G. Such icon semantics can be implicitly defined. Or we can explicitly define icon semantics, by associating an iconic operator INT_j with each rule rj in R. The evaluation procedure is described below.

```
procedure EVAL(xo)
begin
  exp = xo;
  while there is an icon x in exp
        and rule rj is x ::= (Xm,Xi)
    replace x in exp by INTj(Xm,Xi);
end
```

For example, head icon xo of Example 1 of Section 3 can be evaluated as follows:

```
1) xo
2) INT1({x1,x2}, po)
3) INT1({INT2({x3,x4}, p1),x2}, po)
4) INT1({INT2({INT4({c1}, p3),x4}, p1),
            x2}, po)
5) INT1({INT2({INT4({c1}, p3),INT5({c2},
            p4)}, p1), x2}, po)
6) INT1({INT2({INT4({c1}, p3),INT5({c2},
            p4)}, p1),INT3({},p2)}, po)
```

```
If all the INTi's are identical, we have
```

```
7) INT({INT({INT({c1}, p3),INT({c2}, p4)},
            p1),INT({},p2)}, po)
```

Evaluation of the above expression depends upon the INT operator. For the image database of Example 1 of Section 3, INT may be defined as follows.

```
Procedure INT(A, B)
begin
  draw physical image B to scale;
  while there is another element u in set A
  begin
    if u is of the form INT(C,D)
       then INT(C,D)
       else draw logical object u to scale
       within image B;
  end
end
```

This INT operator creates an *image icon* ({}, image) using the above recursive procedure. It will first draw image po. Within po, the sub-regions for p1 and p2 will be redrawn with greater detail. Within p1, the sub-regions for p3 and p4 will be redrawn with even greater detail. Moreover, p3 will be marked with label "c1", and p4 will be marked with label "c2".

In iconic indexing, we can use the generic iconic indexing operator IDX_j to replace the INT_j operator, so that we can reduce a complex icon specified by a formal iconic system into a simpler icon. This new icon can then serve as an index to the original complex icon. Continuing with the above example, the expression in (6) can be replaced by,

8) IDX1({IDX2({IDX4({c1}, p3),IDX5({c2}, p4)}, p1),IDX3({},p2)}, po)

The general form of the indexing operator IDX has been defined in sub-section (9) of Section 4. We now give a specific indexing operator IDX_j, which will behave differently for different rule-index-set.

```
IDXj(A,B)
begin
  if j is in rule-index-set
    then return(union of logical part of
                icons in A, B)
    else return({}, e)
end
```

With this indexing operator, suppose the rule-index-set is {1}, then we will reduce the above expression (8) into an icon: ({}, po). In other words, a low-resolution image po is used as an index. If the rule-index-set is {1,2,3,4,5,6}, then the indexing icon becomes: ({c1,c2}, p0). In other words, the index has a logical part which is a set of subicons (in this case, a set of two ship-like objects), and a physical part which is a low-resolution image po.

Now we can generalize the concept of similarity among icons. Suppose X is a complex icon, and Y is a simpler icon. The two icons are similar, if SIM(IDX(X),Y) is true. The icon Y can also be considered to be an index of X. In icon-oriented information retrieval, we can use Y to retrieve similar icons in the set {W: SIM(IDX(W),Y) is true}.

It can be seen that iconic indexing is a powerful tool to construct indexes from a formal iconic system. For a different application domain, we can define the appropriate indexing operators IDX_j, to create a specific indexing technique.

7. Specification of User Interfaces as Iconic Systems

The examples presented in Section 3 illustrate complex objects such as an image database, a book, or a form, can be specified as iconic systems. In Section 4, we introduced iconic operators. The icons used in direct manipulation interfaces are then seen to be complex icons constructed from elementary icons using iconic operators. The concept of generalized icons therefore has wide applicability. In this section, we will demonstrate that any type of menu-driven user interfaces can also be specified as iconic systems. This requires the use of both object icons and iconic operators.

In a simple menu-driven user interface, there is only one menu page, as illustrated in Figure 1.6.

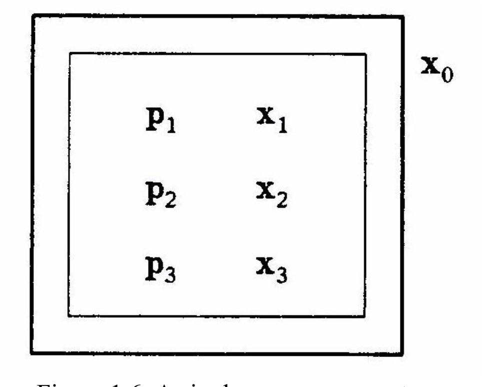

Figure 1.6. A single-page menu system.

The formal iconic system G1 for this menu page is given by:

```
x0 ::= ({x1,x2,x3}, "P1"^"P2"^"P3")
x1 ::= (P1, "P1")
x2 ::= (P2, "P2")
x3 ::= (P3, "P3")
```

where the notation "Pi" denotes the actual image of Pi, and the operator ^ is the vertical concatenation operator to vertically concatenate two images.

The following iconic operators are defined:

SELi (x0) = xi for i=1,2,3, selects a subicon xi in menu icon x0,

EXEC (xi) for i=1,2,3, invokes and executes process pi.

Therefore, x1, x2 and x3 are process icons, and x0 is a complex icon. The user can first select an icon from the menu x0 (using the SELi operator). The EXEC operator can then be applied to invoke and execute the selected process. Conceptually, the iconic operator EXEC is a unary operator, returning a null icon.

A more complex, two-page menu system is illustrated in Figure 1.7.

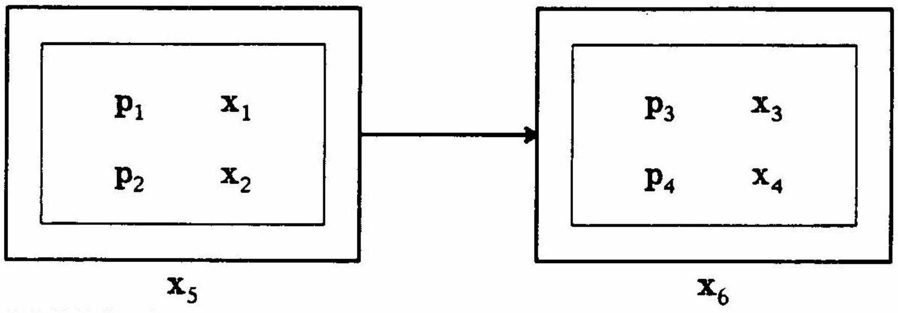

Figure 1.7. A two-page menu system.

The formal iconic system G2 for the two-page menu system is given by:

```
x0 ::= (CONNECTm({x5,x6}), CONNECTi({x5,x6}))
x5 ::= ({x1,x2},"P1"^"P2")
x6 ::= ({x3,x4}),"P3"^"P4")
x1 ::= (P1, "P1")
x2 ::= (P2, "P2")
x3 ::= (P3, "P3")
x4 ::= (P4, "P4")
```

The following iconic operators are defined:

CONNECT({X,Y}) connects two object icons X and Y. $\mathrm{CONNECT}_m$({X,Y}) gives the logical relationship that icon Y is after icon X, and $\mathrm{CONNECT}_i$({X,Y}) is the sketch with two icon sketchs for X and Y connected by a directed arc from X to Y.

FIRST(X) gives the first subicon in the complex icon X.

NEXT(X) gives the next icon which is logically connected to X.

SEL_i(X) selects the i^{th} subicon in complex icon X.

As before, the iconic operators can operate on the icons of this formal iconic system G2 to realize the menu-driven user interface. Other types of menu-driven user interfaces can be specified similarly.

8. Determination of Icon Purity

In this section and the next, we turn to some theoretical considerations which are necessary for a deeper understanding of the behavior of icons in an iconic system.

In Section 3, we introduced a formal approach to specifying iconic systems. Iconic operators were presented in Section 4, which constitute an icon algebra to construct complex icons and define icon semantics. We also introduced the notion of pure icons. Only for pure icons can the logical part be completedly recovered from the physical part, and vice versa. Impure icons may cause problems when used for person-machine communication. In this section, we give purity-preserving conditions for iconic operators.

An icon is denoted by x, its formal identifier, or (X_m, X_i), where X_m is subset of $VL \cup S$, and X_i is an element of VP. We will use the notations x, (X_m, X_i), $x(X_m, X_i)$, and (x, X_m, X_i), interchangeably, to denote an icon. Given an iconic system G, we can determine the iconset RR, which is the set of all icons defined by G, or formally,

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

Let WL denote the power set of VL S, or the set of meanings. The *materialization* and *dematerialization* functions can now be defined.

The materialization function MAT is a partial function from WL to 2^{VP}, defined as follows. For every X_m which appears in (x, X_m, X_i) of RR,

$$\mathrm{MAT}(X_m) = \begin{cases} \{ X_i : (x, X_m, X_i) \in RR \} \\ \text{undefined for other } X_m \end{cases}$$

The dematerialization function DMA is a partial function from VP to the power set of WL, defined as follows. For every X_i which appears in (x, X_m, X_i) of RR,

$$\mathrm{DMA}(X_i) = \begin{cases} \{ X_m : (x, X_m, X_i) \in RR \} \\ \text{undefined for other } X_i \end{cases}$$

An icon (X_m, X_i) is pure, iff

$$\begin{cases} MAT(X_m) = \{ X_i \} \\ DMA(X_i) = \{ X_m \} \end{cases}$$

For elementary icons, it is easy to determine directly their purity from MAT and DMA. An example follows.

Example 4: The icon set RR1 is

$$\left\{ \begin{array}{l} (x_0, \ \{x_1, x_2 \}, \ p_0), \\ (x_1, \ \{x_3, x_4 \}, \ p_1), \\ (x_2, \ \{\}, \ p_2), \\ (x_3, \ \{ c_1 \}, \ p_3), \\ (x_4, \ \{ c_2 \}, \ p_4), \\ (x_5, \ \{ c_2 \}, \ p_5) \end{array} \right\}$$

The MAT and DMA functions are:

$$\begin{cases} MAT(\{ x_1, x_2 \}) = \{p_0 \} \\ MAT(\{ x_3, x_4 \}) = \{p_1 \} \\ MAT(\{ c_1 \}) = \{p_3 \} \\ MAT(\{ c_2 \}) = \{p_4, p_5 \} \end{cases} \qquad \begin{cases} DMA(p_0) = \{ \{ x_1, x_2 \} \} \\ DMA(p_1) = \{ \{ x_3, x_4 \} \} \\ DMA(p_2) = \{ \} \\ DMA(p_3) = \{ \{ c_1 \} \} \\ DMA(p_4) = \{ \{ c_2 \} \} \\ DMA(p_5) = \{ \{ c_2 \} \} \end{cases}$$

Icons x_3 is pure, while icons x_4 and x_5 are not. The icon, x_2, is an image icon. It is pure in this example, but if the iconic system has another image icon, x_2 will no longer be pure. Since image icons are physical images without any label, we usually do not think of them as pure icons.

If we use the above partial functions MAT and DMA, then the structural icons x_0 and x_1 are also pure. Again, we usually do not think of structural icons as pure icons, because they normally represent iconic *relations.*

We now deal with composite icons and their purity. A composite icon $(x, X_m = \{ OP, y_1, \dots, y_n \}, X_i)$ is composed from subicons $y_1, \dots, y_n$ as follows:

$$\begin{cases} X_m = OP_m (y_1, \dots, y_n) \\ X_i = OP_i (y_1, \dots, y_n) \end{cases}$$

where $OP(y_1, \dots, y_n)$ is an n-ary iconic operator.

Two conditions for *purity-preserving* composition of composite icons can now be stated:

$$MAT(OP_m(y_1, \dots, y_n)) = \{ OP_i(MAT(Y_{m_1}), \dots, MAT(Y_{m_n})) \}$$

$$DMA(OP_i(y_1, \dots, y_n)) = \{ OP_m(DMA(Y_{i_1}), \dots, DMA(Y_{i_n})) \}$$

Condition (C-1) says that $MAT(X_m)$, or the materialization of X_m, is equal to the application of the operator OP_i on the individual materialization of the subicons $Y_{m_1}, \dots, Y_{m_n}$. The condition (C-2) can be interpreted similarly.

The consequence of these purity-preserving conditions is stated in the next theorem.

Theorem:

If $y_1, \dots, y_n$ are pure icons, and the above purity-preserving conditions hold, then the composite icon x is also pure.

Proof:

We need to show MAT(X_m) = {X_i} and DMA(X_i) = {X_m}.

$$
\begin{aligned}
MAT(X_m) &= MAT(OP_m(y_1,\cdots,y_n)) \\
&= \{OP_i(MAT(Y_{m,1}),\cdots,MAT(Y_{m,n}))\} && (\textit{because of C-1}) \\
&= \{OP_i(\{Y_{i,1}\},\cdots,\{Y_{i,n}\})\} && (\textit{because } Y_i \textit{ are pure}) \\
&= \{X_i\}
\end{aligned}
$$

Similarly, we can show DMA(X_i) = {X_m}.

Structural icons can be regarded as composite icons with an implicit operator STC. Therefore, if $(x, X_m = \{y_1,\cdots,y_n\}, X_i)$ is a structural icon, we have

$$
\begin{aligned}
X_m &= STC_m(y_1,\cdots,y_n) = \{y_1,\cdots,y_n\} \\
X_i &= STC_i(y_1,\cdots,y_n) = STC_i(Y_{i,1},\cdots,Y_{i,n})
\end{aligned}
$$

For composite icon (x, X_m, X_i), the X_m part is formally denoted by {OP, $y_1,\cdots,y_n$}. By that we mean $OP_m(y_1,\cdots,y_n)$, but we will formally write {OP, $y_1,\cdots,y_n$}. If $OP_m(y_1,\cdots,y_n)$ really generates a new meaning, then the composite icon x becomes once more an elementary icon, i.e.

$$(x, X_m, X_i) = (x, OP_m(y_1,\cdots,y_n), OP_i(y_1,\cdots,y_n))$$

If the purity conditions hold, this newly composed icon is also a pure icon. The implications of the purity-preserving conditions are the following:

(C-1)
: If we find the meanings of all sub-images, then we can combine them to find the meaning of the whole image;

(C-2)
: If we find the images of all the partial meanings, then we can combine them to find the image of the whole meaning.

It should be noted that the purity-preserving conditions also imply:

$$
\begin{cases}
OP_m(y_1,\cdots,y_n) = OP_m(y_{m,1},\cdots,y_{m,n}), \\
OP_i(y_1,\cdots,y_n) = OP_i(y_{i,1},\cdots,y_{i,n}).
\end{cases}
$$

In other words, the operator $OP(y_1,\ldots,y_n) = (OP_m(y_{m,1},\cdots,y_{m,n}),\ OP_i(y_{i,1},\cdots,y_{i,n}))$.

In the above, we defined the purity of individual icons. Can we give an overall measure of the degree of purity of a formal iconic system? This should tell us how "unambiguous" is this iconic system. As we will present in the following section, the purity of a formal iconic system can be defined by extending the iconic system into a fuzzy iconic system, based upon the theory of fuzzy sets.

9. Fuzzy Iconic Systems

A *fuzzy iconic system* is a hexatuple:

$$FG\ (VL, VP, S, x_0, R, M)$$

where $G\ (VL, VP, S, x_0, R)$ is a formal iconic system, and M is the fuzzy membership function from R into [0,1].

Example 5: The iconic system G of Example 4 of the previous section can be extended to a fuzzy icon system FG by associating a fuzzy membership grade with each rule in R1.

$$
\begin{cases}
r_1: & x_0 ::= & (\{x_1, x_2\}, & p_0), & 0.8 \\
r_2: & x_1 ::= & (\{x_3, x_4\}, & p_1), & 1 \\
r_3: & x_2 ::= & (\{\ \}, & p_2), & 0.6 \\
r_4: & x_3 ::= & (\{c_1\}, & p_3), & 0.7 \\
r_5: & x_4 ::= & (\{c_2\}, & p_4), & 1
\end{cases}
$$

R1 can be divided into two sets:

(a)
: Those rules which define elementary icons could be called elementary rules, such as r_3, r_4, r_5.

(b)
: Those rules which define complex icons could be called generative rules since they usually associate subicons with a complex icon, such as r_1, r_2.

The *grade of membership* of the icon X generated by FG is defined as

$$\mu_{FG}(X) = \sup [\min [M(r_1'), \ldots , M(r_m')]]$$

where $r_1', \cdots , r_m'$ are rules used for generating X and the supremum is taken over all derivations of X by FG.

A *fuzzy icon A* in FG is written as

$$A = (A_m, A_i, A_\mu)$$

where

$$A_\mu = \mu_{FG} (A_m, A_i).$$

is the membership grade.

The materialization operator MAT is a fuzzy function defined as

$$MAT(L_m) = \{ P_i, \quad \mu_{FG}(L_m, P_i) \}$$

where L_m mem WL, P_i mem VP, and $WL = 2^{VL \cup S}$. Therefore, $MAT(L_m)$ is a fuzzy set over VP.

Similarly, the dematerialization operator DMA is a fuzzy function defined as

$$DMA(P_i) = \{ L_m, \quad \mu_{FG}(L_m, P_i) \}$$

where L_m mem WL, P_i mem VP. Therefore, $DMA(P_i)$ is a fuzzy set over WL.

The *intension function* of a fuzzy set $A = \{ x, \mu_A(x) \}$ is defined as

$$I_A(x) = \mu_A(x) * (1 - \mathrm{avg}[\mu_A(y) \mid \quad y \neq x]).$$

For convenience, we have the notations

$$I_{MAT}(X_m, X_i) = I_{MAT(X_m)}(X_i)$$

and

$$I_{DMA}(X_m, X_i) = I_{DMA(X_i)}(X_m).$$

We then have:

$$\begin{cases} I_{MAT}(X_m, X_i) = \mu_{FG}(X_m, X_i) * (1 - avg[\mu_{FG}(X_m, X_j) \mid \quad X_j \neq X_i]) \\ I_{DMA}(X_m, X_i) = \mu_{FG}(X_m, X_i) * (1 - avg[\mu_{FG}(X_n, X_i) \mid \quad X_n \neq X_m]) \end{cases}$$

The *degree of purity* of an icon $X = (X_m, X_i)$ can be defined as

$$p(X) = [I_{MAT}(X_m, X_i)]^{K_{MAT}} \times [I_{DMA}(X_m, X_i)]^{K_{DMA}}$$

where K_{MAT} is the *care-factor* for materialization and K_{DMA} is the *care-factor* for dematerialization. Both of them are non-negative numbers.

An icon $X = (X_m, X_i)$ is pure if and only if $p(X) = 1$. In other words, $MAT(X_m) = \{ 1/X_i, 0/\text{others} \}$ and $DMA(X_i) = \{ 1/X_m, 0/\text{others} \}$.

In general, we care more about dematerialization than materialization. That is to say, if the mapping from WL to VP is one-to-many (there are several pictures associated with a single meaning), it is still acceptable. However, if the mapping from WL to VP is many-to-one (there are several meanings for one picture), it is usually not acceptable. Therefore, we can set K_{MAT} to be a small number such as 1 or 2, and K_{DMA} to be a large enough number such as 9 or 10. If care-factor K_{MAT} is 0, this means the intension function for materialization can be ignored. On the other hand, if care-factor K_{DMA} is infinite, this means we can only accept an intension function value of 1 for dematerialization, when the icon is pure.

Given a fuzzy iconic system FG, the degree of purity of FG can be defined as either

$$p_{worst}(FG) = \inf [p_{FG}(X) \mid \quad X \text{ generated by } FG]$$

or

$$p_{avg}(FG) = \mathrm{avg} [p_{FG}(X) \mid \quad X \text{ generated by } FG].$$

The first definition gives the the worst case in such a system, and the second definition gives the average behavior of such a system.

With the above definitions, we can apply the purity measurement on any iconic system G as follows. We first assume a constant fuzzy membership function, i.e., $M(r_i) = 1 \quad FA \; r_i$ mem R. Thus the iconic system G is extended to a fuzzy iconic system FG.

With given values of K_{MAT} and K_{DMA}, we can then calculate the degree of purity of FG, which is also the degree of purity of G.

Example 6: The following iconic system has six elementary icons:

$$\begin{cases} r_1: & x_1 ::= (L_1, P_1) \\ r_2: & x_2 ::= (L_2, P_2) \\ r_3: & x_3 ::= (L_3, P_3) \\ r_4: & x_4 ::= (L_1, P_4) \\ r_5: & x_5 ::= (L_1, P_5) \\ r_6: & x_6 ::= (L_2, P_6) \end{cases}$$

with $WL = \{ L_1, L_2, L_3 \}$, and $VP = \{ P_1, P_2, P_3, P_4, P_5, P_6 \}$.

The icon x_3 is pure, and all the other icons are impure.

If $K_{MAT} = 1$ and $K_{DMA} = 4$, we have:

Icons	I_{MAT}	I_{DMA}	*Purity*
(L_1, P_1)	0.6	1	0.6
(L_2, P_2)	0.8	1	0.8
(L_3, P_3)	1	1	1
(L_1, P_4)	0.6	1	0.6
(L_1, P_5)	0.6	1	0.6
(L_2, P_6)	0.8	1	0.8

Therefore,

$$P_{avg}(FG) = 0.73.$$

If we replace r_6 by

$r_6: x_6 ::= (L_3 , P_2)$

we have:

Icons	I_{MAT}	I_{DMA}	*Purity*
(L_1, P_1)	0.5	1	0.5
(L_2, P_2)	1	0.5	0.0625
(L_3, P_3)	0.75	1	0.75
(L_1, P_4)	0.5	1	0.5
(L_1, P_5)	0.5	1	0.5
(L_3, P_2)	0.75	0.5	0.046875

Therefore,

$$P_{avg}(FG) = 0.39$$

The icons are all impure, and the average degree of purity is also less than that of the previous iconic system. However, it is not zero.

The nice thing about the above definition of degree of purity is that *there is no need to know the fuzzy membership function of an iconic system.* We can always assume a constant fuzzy membership function, and we are still able to compute a meaningful average degree of purity.