An Implementation of a Model for Functional Recognition

Marie-Christine Jaulent
Service d'Informatique Médicale, Hôpital Broussais, 96 rue Didot, 75014 Paris, France

Lucia M. Vaina
Intelligent Systems Laboratory, College of Engineering, Boston University, Boston, Massachusetts 02215

This article addresses the problem of the recognition of the usefulness of man-made objects in actions. In a recent article, [L. M. Vaina and M. C. Jaulent, *Intelligent System Journal* **6**, 313–336 (1991)] we defined the functional recognition problem in terms of the compatibility between the primary functions of objects and action requirements. We proposed a model, based on fuzzy sets and possibility theory, for the organization of this knowledge and the evaluation of the compatibility to perform functional recognition. In this article, we present an implementation of functional compatibility which takes the functional descriptions of objects and the object requirements of an action as inputs, and maps them into an output space that evaluates their functional compatibility.

I. INTRODUCTION

Broadly speaking, visual object recognition involves matching the description of an object with previously stored items in memory. Popular forms of visual object recognition methods are the *hypothesis–prediction–verification* paradigm,[2] the *constrained search scheme*,[3] the *relaxation scheme*,[4] and the *rule based production scheme*[5]. Each of these models is concerned with the construction of explicit descriptions of physical objects from their images. Such descriptions, however, may not be sufficient when the purpose is object manipulation rather than object recognition. We proposed[1,6] that, for manipulation, objects must also be described by their functions or possible uses in actions. Moreover, we argued that for tasks of manipulation and reasoning about objects, structural and geometrical descriptions of objects in a model are not sufficient for discrimination and recognition of the possible uses of objects in actions.

As d'Arcy Thompson[7] put it, "form follows function," and this is especially true of the man-made objects in our every-day environment. In order to be

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 9, 379–402 (1994)

CCC 0884-8173/94/040379-24

useful in actions, the object's structure must have certain properties designed to realize those actions. Whereas other approaches to functional representation have addressed the problem of object design, our focus is on the recognition of functions.

In a representation of the functional dimensions of objects, the functional properties of objects are explicitly expressed (Fig. 1). We restrict our study to the functional recognition of manufactured objects useful in hand actions.

Functional information is an intrinsic part of one's general knowledge about objects: the "object-concept."[1] The relation between the structure and function of objects is often not directly available and must be extracted through reasoning. From a computational point of view, this means that the functional information is only implicit in the perceptual representation of an object, and computations are required to make it explicit.[6]

In a previous study, we identified a set of constraints on the functional representation and proposed a theoretical model of functional compatibility between manufactured objects and hand actions[1] using the framework of possibility theory.[8]

The present study contributes to the computational validation of this theoretical model by first addressing the constraints on the representations involved in the model, and then describing a specific algorithm that evaluates the functional compatibility between objects and actions.

II. A MODEL FOR FUNCTIONAL RECOGNITION

A. General Objective

The general objective of the functional recognition task is to associate objects and actions. However, objects and actions cannot be compared directly. Rather, they must be released through *functions*.

In daily human activities, functional or practical knowledge is invoked in order to determine whether an object can be used in an action or whether an object can serve a certain purpose (the goal of the action). For example, if one wants to drive a nail into a board, the natural thing to use is a hammer. This use of the hammer we define as the *primary function* of an object: the specific action for which the object has been manufactured (we may know this from the lexical or categorical knowledge domain). However, we would also know that a rock, a certain part of pliers, or in some circumstances, the handle of a screwdriver may accomplish the same goal. This sort of knowledge is different in nature, it is derived through inferences on the physical properties of these objects and the requirements of the action to *pound* (*a nail*). This additional function of an object, beyond its primary function, is called an *auxiliary function*. The task or functional recognition concerns both primary and auxiliary functions.

In addressing the problem of functional recognition, we must first decide which representations to use, and then devise and implement an algorithm

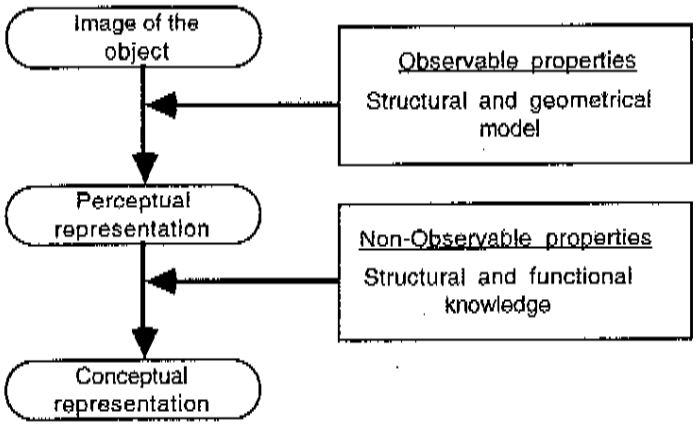


Figure 1. The place of the functional dimensions in the successive representation of an object.

which evaluates the functional compatibility between descriptions of objects and action requirements.

B. Constraints on the Representations

The goals of correctness, efficiency, and robustness of a functional recognition system place many constraints on its design and performance, especially on the nature of the representations that may be used.

1. The Conceptual Representation

Functions relate objects and actions and this relationship is mediated through object parts and their configuration.[1,9,10] The structure and description of parts form the appropriate level of abstraction for the representation of object functions. Consequently, the conceptual representation (Fig. 1) must make explicit the direct association between the structural components of objects (parts) and their functional properties (as defined in the primary function of the object). For instance, a knife has two structural parts, the blade and the handle. The blade directly implements the action while the handle facilitates the implementation of the action. The handle affords *grasping* (the associated functionality) of the object while the blade affords its specific use for *cutting* (the associated functionality). Some authors[11] have argued that the functional descriptions of a wide set of manufactured objects can be approached by decomposing them into networks of functional primitives. This approach requires detailed knowledge of the constraints imposed on shapes by functions and defines a limited number of functional primitives for associating functions with shapes.[9] Our approach is different in that our conceptual representation (the functional description) makes explicit only the primary function of the object. The general function primitives are not explicitly attached to any specific manufactured object, they

are part of the general commonsense knowledge about functions expressed in terms of physical properties rather than in terms of specific shapes.

2. *The Functional Knowledge Representation*

The invariant physical properites and relations which can fulfill a function constitute the *functional prototype* of that specific function. It explicates the association between a singular functionality and its structural, geometrical, and descriptive properties.

It is important to note that the functional prototype is not an ideal object with a particular function or a primary function. There is a fundamental difference between a prototype and a primary function. For example, "to-screw" is a functional prototype and not the primary function of a given screwdriver. We consider the functional prototype as a more generic concept which can be afforded by a class of objects with possibly different primary functions or even no primary function at all (e.g., coin, knife, screwdriver-1, screwdriver-2, etc.). The primary function of a given screwdriver is: "to drive a screw of a precise length, a precise shape through a specific movement of the hand."

C. Object Representation

Vaina[12,13] proposed a model for representing commonsense knowledge in terms of three modules: categorical, conceptual, and perceptual.

The *object-category* module is a hierarchical representation of objects organized in terms of their semantic categories. Thus, for example, a cup is an object, man-made, able to be manipulated, and a container. The role of this representation is to provide classification and taxonomies of objects.

The *object-concept* module, a conceptual representation, makes explicit the object's parts, their relations, and functions. Parts which occur at the same level of detail (e.g., the legs of a chair) are described at the same level in the hierarchy. In addition to the description of parts and relations, the object-concept level also makes explicit the specific function associated with the object, usually by a direct link between a part and the function that it affords. Only the primary function is made explicit in the object concept.

The *object-structure* module is the perceptual representation whose role is to make explicit the structure of the object. It describes the shape of the parts, their geometrical relations, and their visual attributes. Naturally, one would prefer a general purpose shape recognizer which does not impose too strong demands on the earlier processes. In essence, it would be desirable to have a system which could handle equally well the recognition of pineapples, tomatoes, cats, cups, and hammers, even though in certain circumstances some of these objects might be better described by "special purpose" representations. However, this is not practically feasible. The structural characteristics of objects, the nature of the regularities discovered by the earlier vision mechanisms, will constrain the representation that could provide an appropriate description of a three-dimensional object and its parts. As discussed in the introduction,

for the purpose of recognizing the possible uses of objects in actions, it is necessary to rely on shape representation for the visual input which can give accurate descriptions of parts effortlessly and quickly.

Three main methods have been proposed for isolating parts. The first method is usually referred to as *axis-based*[14,15] and relies on the decomposition of objects into parts based on the axes of symmetry. The second method is called *primitive-based,* and it relies on defining the possible shapes (most commonly cylinders, spheres, cones or polyhedras) of the parts. These two methods each have their advantages and disadvantages. The axis-based method is very useful for objects which have axes of symmetry and when the axes are easily accessible from the earlier processes, while the primitive based method is very useful for a priori specified classes of objects, such as man-man objects. Once one decides on the appropriate primitive parts, the task is to locate these parts in the object and associate them with characteristic metrical properties (e.g., length) and then relate them by predicates which express spatial relations (e.g., to the right of). Many of our recognition tasks in daily life are facilitated by context or by the expectations of the kinds of objects we may be dealing with. Specifically, this holds quite well in the domain of computer vision, for which most of the part-based methods have been developed.

The third method is *boundary-based,* and in this method parts are defined through their boundary with adjacent parts.[16-20] Both the primitive-based and the boundary-based methods have in common the fact that they define parts on a representation of the object which is independent of the vantage point and fully specifies the 3-D shape (the volume occupied by the object) or, equivalently, the whole 3-D bounding surface. Bennett and Hoffman[18] have demonstrated that boundary-based methods do give a part definition which is completely general. In addition, we argue that the boundary-based methods are well suited for discovering how to use an object in both its primary and auxiliary functions.[1] This is due to the fact that one visually samples not the whole of an object, but the parcellated object,[21] and for the perceiver an object is first defined as the visual response to its parcellation which is obtained through exploratory movements. Hence we argue that one first obtains a *functional* definition of the object rather than a *description* of it.

D. Action-Requirements Representation

The necessary and sufficient properties for achieving the goal of an action are collectively called action-requirements. They involve *movement-requirements* which refer to how the action is carried out, and *object-requirements* which embody the required functional properties of the object.[1] The object-requirements provide constraints on the structure of the object in order for that object to be useful in an action. Thus, to be useful in an action such as digging, the two necessary functions of "containing" and "cutting" must be fulfilled by the same structural part. We will focus here only on the object-requirements (but will use the term *action-requirements*).

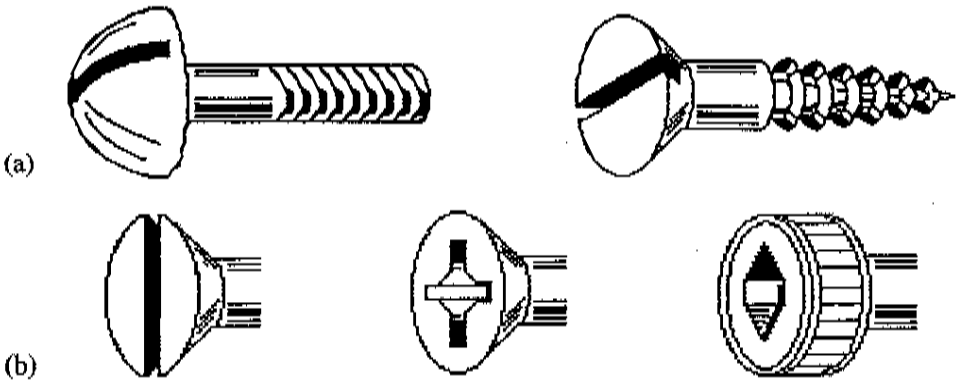


Figure 2. (a) Different screws to drive different materials. (b) Different screws requiring different screwdrivers.

The representation of a set of action requirements must include information about the context in which the object is used. Although the functions "to cut butter" and "to cut a log" both require the same physical property of "a sharp edge," the sharpness of the edge must differ in the two actions, resulting in two different sets of action requirements. The context constrains the domain of admissible values for a physical property and is embedded in a space of reference.

To illustrate, Figures 2(a) and 2(b) show different types of screws. Figure 2(a) shows screws differing on the basis of the material in which they are used (wood, metal, etc.). These differing media imply different structural requirements on the part that is driven (by a screwdriver) and the part that is grasped. The requirements are delineated by the force necessary to drive the screw. Figure 2(b) shows different screw heads which impose different functional requirements on the shape of the screwdriver head. Depending on which screw one wants to use, the action-requirements to perform the action "to-screw-that-screw" will be different.

E. A Compatibility Model

We defined the functional recognition problem[1] as the problem of finding the functional compatibility between object requirements and object properties.

The evaluation of functional compatibility consists of finding a measure of similarity between the set of those properties required by a specific action (objects-requirements) and the description of an object in the three-modules representation (Fig. 3). The model is based on a pattern-matching procedure between the two inputs of the action requirements and the object's description.

1. A Pattern-Matching Procedure

Pattern matching is a tool that has many applications in computer vision which allows an interpretation of the input data (visual input) to predefined patterns (models).[22,23] The interpretation is performed by relating visual discriptions to a catalogue of models.[24]

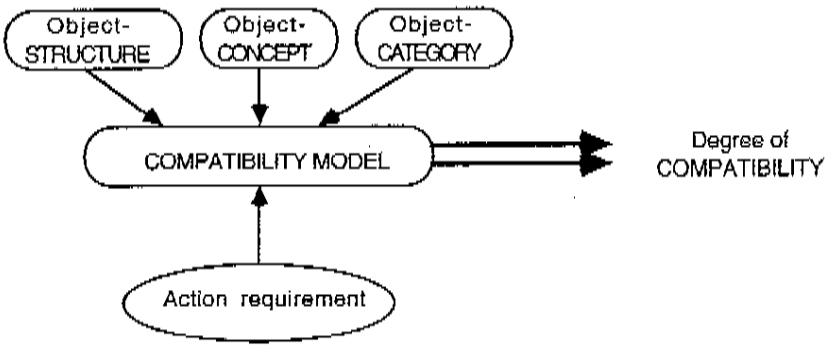


Figure 3. Outline of the compatibility model.

The first step of a pattern-matching procedure is to verify that the two entities are described by the same basic primitives.[23] In the three modules representation of the object, each module has its own set of descriptors. Consequently, we define the pattern-matching procedure to be a three-step process: *category pattern matching*, *conceptual pattern matching*, and *structural pattern matching*. Category pattern matching refers to Rosch's basic category concept[25] and verifies that the candidate for a match can be held by the hand.

The second step of a pattern-matching procedure is to define a support to express the "resemblance" between the two entities. Matching involves quantifiable similarity which evaluates resemblance and this can be expressed by minimizing dissimilarity or maximizing similarity,[26] by logically combining matching and mismatching properties[27] or by using ad-hoc metrics.[28] Tversky's contrast similarity model is particularly interesting here, since the similarity between two objects depends not only on the features they have in common, but also on the features by which they differ. Tversky has demonstrated that categorical similarity is not psychologically symmetric and that the metrics of a similarity measure is strongly information-dependent.

2. *Pattern and Data*

The set of action-requirements defining an action and the appropriate spaces of reference generate the conceptual pattern of the procedure, while a functional property generates the structural pattern [Fig. 4(a) and Fig. 4(b)].

Both the conceptual and structural representations of an object form the data of the procedure.

3. *Outline of the Procedure*

The pattern-matching procedure operates initially at the conceptual level. When a "function" is not found directly at this level (independent of the compatibility rate), a new pattern-matching procedure is triggered at the structural level. This operates between the functional prototype associated with the

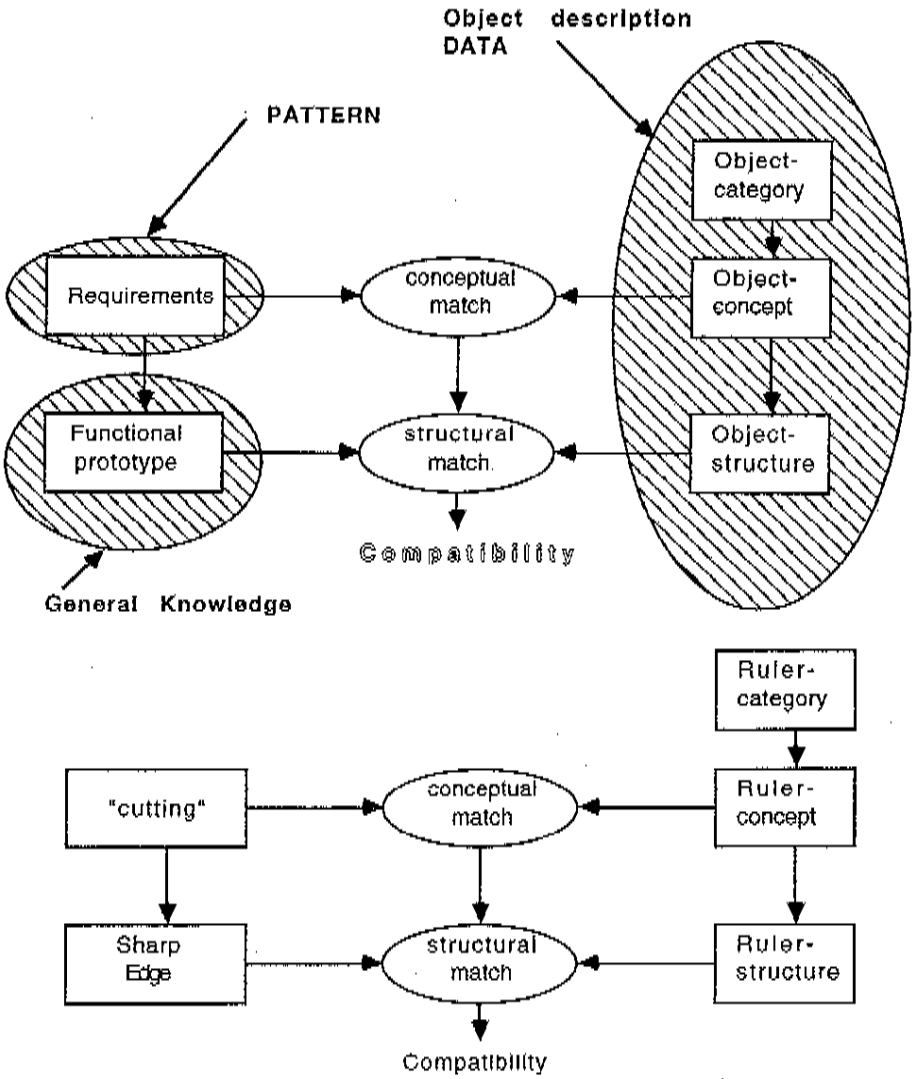


Figure 4. (a) Levels of matching. (b) A "ruler" can cut butter.

missing function and the structural representation of the object. At the structural level, the description of parts must facilitate the extraction of the physical properties.

We now focus on the implementation of a compatibility model. In this model, we will describe objects and actions in terms of functional features. If an object has been especially designed to implement an action (e.g., bread knives are designed for cutting bread, fish knives for cutting fish) the compatibility is obviously trivial: the action requirements are automatically matched by the object functional properties. The compatibility model is problematic when we try to determine the usefulness of objects for actions other than those which directly embody the designer's goal.

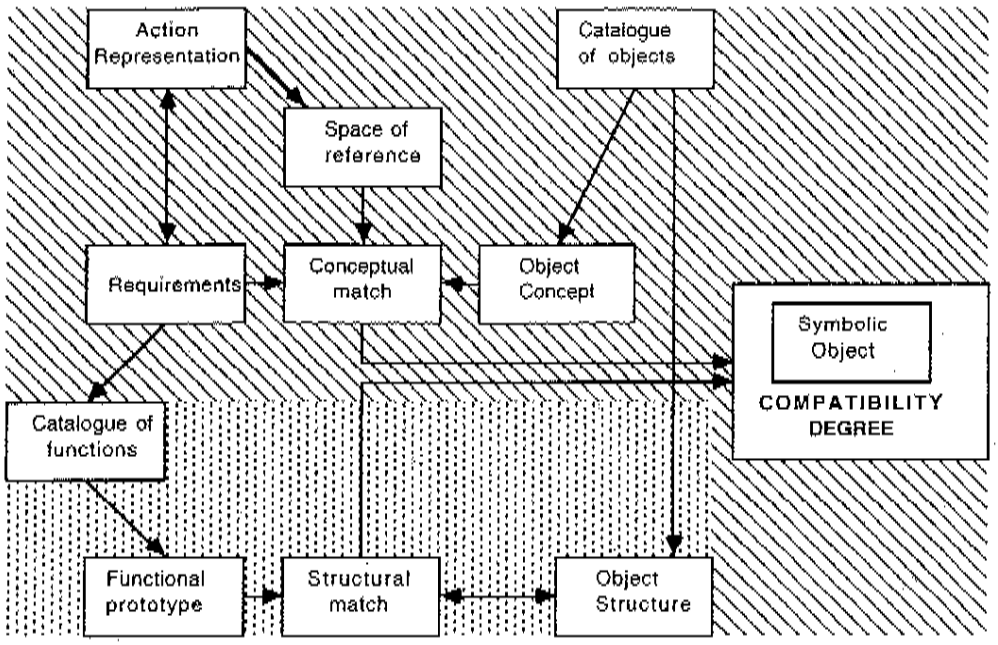


Figure 5. Synoptic representation of the global system.

III. IMPLEMENTATION

The implementation of the compatibility model is composed of several modules. The organization of the different modules is shown in Figure 5. Some of these modules have been fully implemented (hatched area) while others still need to be validated. We will discuss the modules presented in the hatched area.

Globally, the subsystem takes the conceptual representation of one object from a catalogue of objects and a set of action-requirements representing one action and provides: (a) a global compatibility which expresses the extent to which the objects fit the requirements and, (b) a symbolic representation in which the object is described in terms of how to be used in the action.

In the current implementation, when the structural pattern matching is required it is triggered at the user level through an appropriate interface.

Global compatibility reflects an aggregation of all the elementary compatibilities computed for the single requirements. The estimation of the elementary compatibilities and the modeling of the aggregation operations is based on possibility theory.[8,29,30] The global compatibility is expressed by two numerical values: the first value representing the possibility that the object can be used to achieve the action, and the second value being the necessity (the dual measure) expressing the impossibility of the opposite event.[8]

An important aspect of the system is the symbolic representation of the object (also referred to as the goal-oriented representation). This representation is used when the hands actually grasp the object (give the appropriate position)

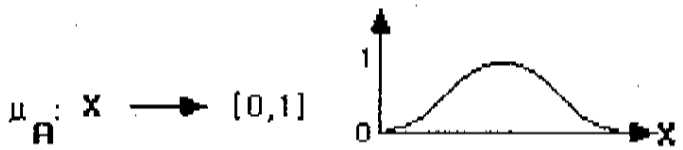


Figure 6. The membership function of a fuzzy set **A**.

and perform the movement underlying the action. This new conceptual representation of the object makes explicit the structural and functional properties of the object compatible with the action requirements. For example, the symbolic representation of "a ruler" for the action "to-cut-butter" will contain only one part (the body part of the ruler) which affords grasping as well as cutting soft material like butter. When the match is successful at the conceptual level, the symbolic representation is a subset of the conceptual representation. Otherwise it is a new representation which makes explicit the specific auxiliary function.

The current system is implemented in Common-Lisp on a Lisp-machine (Gigamos Inc). For increasing efficiency, a subsequent version will be written in C on a DEC Station 5000 under Ultrix.

A. Implementation of the Action-Requirements

Each action-requirement describes a specific functional property afforded by an object ideally suited for the action. A functional property is expressed by a list (ATTRIBUTE (SPACE VALUE)). "Attribute" refers to a functional prototype, "space" refers to the space of reference of the functional prototype for the given action, and "value" is the value required in this particular space. In the example *to pound* (*a nail*), the action-requirement: "to have a flat surface that affords pounding the nail" is expressed by the following list: (POUND (SURFACE LARGER-THAN-NAIL-HEAD)).

The values of an attribute in a given space are often imprecise in the sense that their bounds are not well-defined. For instance, for the space "capacity," when does "deep" become "shallow"? Where is the boundary between the two? We suggest[1] that the fuzzy sets theory[31] allows the representation and the interpretation of an imprecise and subjective meaning of a functional property. Fuzzy sets are appropriate to use in this representation because a fuzzy set is a set for which the membership function takes its values in the interval [0, 1] rather than in the subset {0, 1}. The value 0 corresponds to the absolute nonappartenance and the value 1 corresponds to the absolute appartenance. A fuzzy set **A** in a reference space **X** is entirely defined by its membership function μ_A (Fig. 6).

For all x in **A**, the quantity μ_A (x) evaluates the appartenance of the value x to the set **A** and it is called the membership degree of x in **A**.

Example. Let **A** be the fuzzy set **A** = {x|x is deep} defined on the space

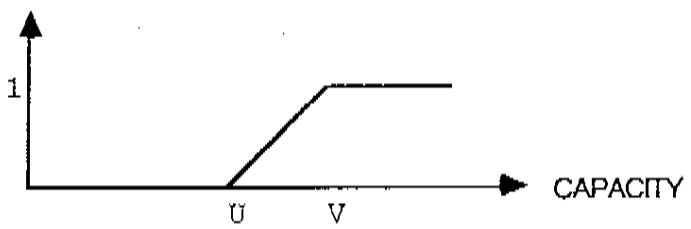


Figure 7. A possible representation of the fuzzy set "DEEP".

"Capacity" (a linear, objective referential, subset of the reals). Figure 7 shows a possible membership function of **A**.

A set of values is associated to each space of reference. A value is entirely defined by a quadruplet, the classical trapezoidal computer representation of a fuzzy set.[29,30] So, for instance:

```
(PUTDROP 'CAPACITY
         '(20 (VERY-SHALLOW 0 6 0 2) (SHALLOW 5 11 1 2)
              (DEEP 10 16 1 2) (VERY-DEEP 15 20 1 0))
         'SPACE-OF-REFERENCE)
```

1. Data Structure Associated to Action-Requirements

We restricted the action to have only two properties, the REQUIRE property and the WEIGHT property. The WEIGHT property will be discussed in Section III-C.3. The REQUIRE property is a collection of functional and structural constraints on the objects. It has a semantic network structure where nodes are functional constraints and links are structural constraints. The latter are of two types: type-part links and connections. A type-part link is used when two required functionalities have to be located specifically in two different parts (two different named TYPE-PART) and connection is used when two functional parts are required to be physically attached. The access to this property is associative.

The following example shows the lisp expressions for the action-requirement representation associated to the action "to screw" when a flared screwdriver is required. In example a, to perform the action does not require a lot of strength and there are no constraints on where the graspable property is located. In example b, some strength is required and the object must have an independent graspable part.

```
a:
(PUTPROP 'TO-SCREW-1
         '(TYPE-OBJECT (GRASPABLE)
                       (SCREWING (SHAPE FLARED) (LENGTH SHORT)))
         'REQUIRE)
b:
```

```
(PUTPROP 'TO-SCREW-2
         '(TYPE-OBJECT (TYPE-PART (NAME X1) (GRASPABLE))
                       (TYPE-PART (NAME X2)
                                  (SCREWING (SHAPE FLARED)
                                            (LENGTH SHORT)))
                       (CONNECTION X1 X2 (R-0 EXTENSION)))
         'REQUIRE)
```

B. Three Modules for Knowledge Representation

The knowledge used by the system to perform the functional recognition is organized into three modules **M1, M2,** and **M3** which form the knowledge representation system.

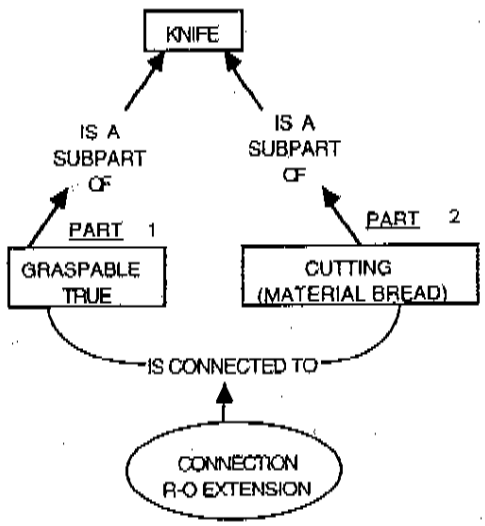
M1 is a catalogue of objects. In the current state of the system, there is no true organization in terms of the categories inside the catalogue. This catalogue is just a collection of objects in which the system searches for a possible object to perform the action. A future extension of the system will be to organize hierarchically this knowledge base in order to preserve the different classes of manufactured objects and the natural inclusions between objects in each class (a Champagne glass is a glass with a stem and all glasses belong to the class of containers). **M1** relates objects to their primary function (if any) and their structural representation.

M2 contains knowledge about functional prototypes and relates functions to their physical properties. The access to this knowledge base is associative; each function is directly associated to its structural and physical properties. We will not discuss here the representation of the structural and physical properties of the functional prototype.

M3 contains knowledge about actions and relates actions to functions. Here again, access is associative. As was pointed out in Section II-D, the knowledge about an action is limited to the knowledge of the structural and functional properties that must afford a manufactured object for successfully performing that action.

1. Data Structure Associated to Objects

An object is an entity (or "atom") with four classes of properties, the STRUCTURE-LEVEL property, the CONCEPT-LEVEL property, the CATEGORY-LEVEL property, and the SYMBOLIC-LEVEL property. Each property is represented as a record containing information specific to that level. The access to these properties is associative and performed by the classical Lisp retrieval function: (GET OBJECT 'STRUCTURE-LEVEL), (GET OBJECT 'CONCEPT-LEVEL), (GET OBJECT 'CATEGORY-LEVEL), (GET OBJECT 'SYMBOLIC-LEVEL). The latter is different in the sense that it is associated to a particular action and accesses the side-effect result of the global pattern matching procedure.




```
(PUTPROP
 'KNIFE
   (MAKE-OBJECT
    :NAME 'KNIFE
    :SON     (LIST                                    PART 1
                (MAKE-OBJECT
                  :DAD 'KNIFE
                  :NAME 'HANDLE
                   :CONNECTION '((BLADE (R-O EXTENSION)))
                   :ATTRIBUT '((GRASPABLE)))
                (MAKE-OBJECT                          PART 2
                  :DAD 'KNIFE
                  :NAME 'BLADE
                   :CONNECTION '((HANDLE (R-O EXTENSION)))
                   :ATTRIBUT '((CUTTING (MATERIAL BREAD)))))))
 'CONCEPT-LEVEL)
```

Figure 8. Graphic and Lisp representation of a KNIFE.

The current version of the system considers the concept-level property of an object as the primer data used in the matching procedure. The value of this property makes explicit the functional parts and their structural relations in a hierarchical structure similar to the Marr and Nishihara model.[24] For instance, parts which occur at the same level of detail (e.g., the legs of a chair) are described at the same level of the hierarchy. The relations are of different types such as, "is connected to" which specifies which parts are attached to each other and "is a subpart of" which breaks down a part in several subparts. Figure 8 shows both the graphic and lisp expression for the conceptual representation of a knife whose primary function is to cut bread.

C. The Pattern-Matching Procedure

The pattern matching between the CONCEPT-LEVEL property of an object and the REQUIRE property of an action occurs in two steps. The system first looks inside the conceptual representation of the object to be extracted, which

is, if possible, a structure similar to the structure required by the action (the structural organization of the action-requirements). This operation is performed by a search procedure, each step of which a partial functional compatibility is computed. One advantage of this algorithm is that it provides a partial result in the case of failure. The symbolic representation of the object reflects the success (or partial success) of the procedure and is built at the same time that the functional compatibility is computed. When a failure occurs during the application of the matching, a backtrack procedure is triggered which will correct the partial functional compatibility and modify the symbolic representation. A global failure will trigger structural pattern matching.

The special variable RESULT-MATCH gives a trace of the matching at any time of the procedure. The module PARTIAL-MATCH is an additional function which stores all the detected partial matchings to reconsider in case of backtracking.

1. Search Procedure

The isomorphic comparisons between the pattern and the datum structures are performed by a simple breadth-first search algorithm.[23] Each functional part at a given level of the representation will be examined before the procedure goes to a deeper level. A breadth-first search algorithm is justified in the context of functional recognition since details are not relevant when recognition occurs at a coarse level. Thus, a common knife might appear suitable to cut a soft material (like butter) without verifying that the teeth of the knife are exactly appropriate to that kind of material.

For a type-part link, the module TYPE-PART looks for a compatible part inside the data which has not been already associated with a previous type-part link of the pattern structure.

For a connection link, the module CONNECTION checks the connections among the parts already named.

For a simple attribute, the module MATCHING-ATTRIBUTE computes the elementary compatibility between an attribute in the pattern and an attribute in the datum.

2. Computation of an Elementary Compatibility

In the current state of the system, the elementary compatibility between two functional properties is evaluated only when the two spaces of reference are the same. Consider for example, the three following functional properties:

```
F1 = (CUT (MATERIAL HARD))
F2 = (CUT (SHARPNESS SHARP))
F3 = (CUT (MATERIAL SOFT))
```

There is certainly a strong relation between the two spaces SHARPNESS and MATERIAL.[1] In the current version of the system, however, this relation is

not made explicit and as a result, only the functional properties F_1 and F_3 can be compared. In this case, the elementary compatibility between F_1 (a functional property from the pattern) and F_3 (a functional property from the datum) is given by possibility Π and necessity $\mathbf{N}$ that the value of F_3 is compatible with the value of F_1,

$$(\Pi(Val_3/Val_1), \mathbf{N}(Val_3/Val_1))$$

3. *Global Aggregation*

Global compatibility is defined by a combination (or aggregation) of all of the elementary compatibilities associated with the action-requirements. The aggregation is performed by the min operation taken over the set of requirements.

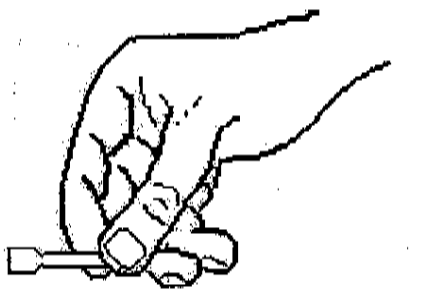
$$\Pi(\text{Action/Object}) = \min \{\Pi(val_i)/F_i\}$$
$$\mathbf{N}(\text{Action/Object}) = \min \{\mathbf{N}(Val_i)/F_i\}$$

This aggregation depends on the salience of each requirement (how much of the functional property can be missing with the object still suitable for performing the action?) as well as the nature of the different links involved (type-part or connections). In order to take into account the difference in salience between the requirements, a weight a_i is associated to each action-requirement F_i reflecting such information with the following convention, $a_i = 0$ indicates that the requirement is negligible while $a_i = 1$ indicates that the requirement is essential. Such information about the action is made explicit in the WEIGHT property of the action. Figure 9 displays the LISP expressions of the WEIGHT property of the action "to-screw" as introduced in Section III-A. For the action TO-SCREW-1, great strength is not necessary, so how and where the screwdriver is grasped is of less importance than the type of screwdriver head used. It is most important that the screwdriver head fits perfectly into the screw head. On the other hand, for the action TO-SCREW-2, the screw head is bigger so the shape of the screwdriver is not as important. In this case, strength is necessary and of greater importance to the design of the object, so the object should have a graspable part.

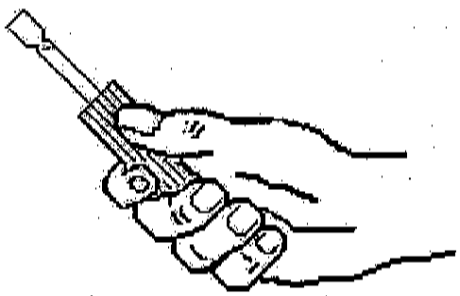
Finally, global compatibility is defined by[1]:

$$\Pi(\text{Action/Object}) = \min \{\max (1-a_i, \Pi(Val_i)) / F_i\}$$
$$\mathbf{N}(\text{Action/Object}) = \min \{\max (1-a_i, \mathbf{N}(Val_i)) / F_i\}$$

The final degree of compatibility is provided by the pair of values (Π(Action/Object), $\mathbf{N}$(Action/Object)) which evaluate the usefulness of the object to perform a given action.



Precision grip



Power grip

Figure 9. Examples of weighting.

IV. CONCLUSION

The work presented in this article is a follow up of our previous more theoretical studies which addressed the problem of the recognition of object functions. Here we present an implementation of our model of functional recognition as the compatibility between objects and their use in hand actions. This model is based on fuzzy sets and possibility theory, where the compatibility is computed by a three-leveled pattern-matching procedure (*category pattern-matching, conceptual pattern matching,* and *structural pattern matching*). We consider two classes of necessary and sufficient properties for achieving the goal of an action. The REQUIRE property is the collection of functional and structural constraints on the objects and the WEIGHT property describes to what extent each REQUIRE property is necessary. These two properties are used to locate within a catalogue of objects those objects that are useful in carrying out the action. The object is described in a three-modules representation (the CATEGORY-LEVEL, the CONCEPT-LEVEL, and the STRUCTURAL-LEVEL) which constitutes the datum of the procedure. The general knowledge of how a single functionality is linked to the structural, geometrical, and descriptive properties of shapes is stored in a catalogue of functional prototypes. The currently implementing system matches the CONCEPT-LEVEL property of the object and the

REQUIRE property of the action and computes the global compatibility between the object and the action. An important secondary component of this procedure builds a symbolic representation of the object: the goal-oriented representation of the object for the given action. In the current implementation, when structural pattern matching is required, it is triggered at the user level through an appropriate interface between the functional prototypes and the STRUCTURAL-LEVEL properties of the object. The proposed conceptual model between objects and their use in hand actions would have many applications due to the fact that it enhances the difference in computations between primary and auxiliary functions. This model, if applied in the area of robotics, could provide an intelligent robot with the capability to carry out a task in an unconstrained environment, and this will be useful especially as unattended manufacturing advances.

V. EXAMPLES

A. A Set of Screwdrivers

SCREWDRIVER-1 and SCREWDRIVER-1-BIS describes the same flared screwdriver except that the parts (the blade and the handle) are not described in the same order and that the blade is graspable in the second one. The results show how the pattern-matching procedure is sensitive to this parameter. SCREWDRIVER-2 describes a torx screwdriver.

```
(PUTPROP 'SCREWDRIVER-1
         (MAKE-OBJECT
          :NAME 'SCREWDRIVER-1
          :SON (LIST   (MAKE-OBJECT
                        :NAME 'HANDLE
                        :CONNECTION '((BLADE (R-O EXTENSION)))
                        :ATTRIBUT '((GRASPABLE)))
                       (MAKE-OBJECT
                        :NAME 'BLADE
                        :CONNECTION '((HANDLE (R-O EXTENSION)))
                        :ATTRIBUT '((SCREWING (SHAPE FLARED)
                                              (LENGTH MEDIUM))))))
         'CONCEPT-LEVEL)

(PUTPROP 'SCREWDRIVER-1-BIS
         (MAKE-OBJECT
          :NAME 'SCREWDRIVER-1-BIS
          :SON (LIST   (MAKE-OBJECT
                        :NAME 'BLADE
                        :CONNECTION '((HANDLE (R-O EXTENSION)))
                        :ATTRIBUT '((SCREWING (SHAPE FLARED)
                                              (LENGTH MEDIUM))))
                       (MAKE-OBJECT
                        :NAME 'HANDLE
                        :CONNECTION '((BLADE (R-O EXTENSION)))
                        :ATTRIBUT '((GRASPABLE)))))
         'CONCEPT-LEVEL)
```

```
(PUTPROP 'SCREWDRIVER-1
         (MAKE-OBJECT
          :NAME 'SCREWDRIVER-1
          :SON (LIST (MAKE-OBJECT
                      :NAME 'HANDLE
                      :CONNECTION '((BLADE (R-O EXTENSION)))
                      :ATTRIBUT '((GRASPABLE)))
                     (MAKE-OBJECT
                      :NAME 'BLADE
                      :CONNECTION '((HANDLE (R-O EXTENSION)))
                      :ATTRIBUT '((SCREWING (SHAPE TORX)
                                            (LENGTH SHORT))))))
         'CONCEPT-LEVEL)
```

B. A Set of Actions

Each action is described along the REQUIRE and PONDERE properties.

In the first example, the two parts are required to be the same. A *WITH* link.

```
a- (PUTPROP 'TO-SCREW-1
            '(TYPE-OBJECT (GRASPABLE)
                          (SCREWING (SHAPE FLARED) (LENGTH SHORT)))
            'REQUIRE)
   (PUTPROP 'TO-SCREW-1
            '((GRASPABLE 0.6) (SCREWING 1))
            'WEIGHT)
```

In this example, the two parts are required to be different. A *APART* link. A connection requirement appears in the prototype.

```
b- (PUTPROP 'TO-SCREW-2
            '(TYPE-OBJECT (TYPE-PART (NAME X1) (GRASPABLE))
                          (TYPE-PART (NAME X2)
                                     (SCREWING (SHAPE FLARED)
                          (LENGTH SHORT)))
                          (CONNECTION X1 X2 (R-O EXTENSION)))
            'REQUIRE)
(PUTPROP 'TO-SCREW-2
         '((GRASPABLE 1) (SCREWING 0.2) (CONNECTION 0.2))
         'WEIGHT)
```

In this case, there is a *WA* link. The connection requirement appears in the implement part of the prototype with a variable >X.

```
c- (PUTPROP 'TO-SCREW-3
            '(TYPE-OBJECT (GRASPABLE)
                          (TYPE-PART (NAME X2)
                                     (SCREWING (SHAPE FLARED)
                                (LENGTH SHORT))
                                     (CONNECTION X2 >X
```

```
(R-O EXTENSION))))
            'REQUIRE)
  (PUTPROP  'TO-SCREW-1
            '((GRASPABLE 0.6)  (SCREWING 1)  (CONNECTION 0.2))
            'WEIGHT)
```

C. Examples of Spaces of Reference

```
(PUTPROP  'SCREWING
          '((FLARED)  (PHILLIPS)  (POZIDRIV)  (REED)  (CLUTCH)
            (ROBERTSON)  (TORX))
          'SHAPE)
(PUTPROP  'SCREWING
          '((VERY-SHORT 0 6 0 2)  (SHORT 5 11 1 2)  (MEDIUM 8 12 1 1)
            (LONG 10 16 1 2)  (VERY-LONG 15 20 1 0))
          'LENGTH)
(PUTPROP  'CONNECTION
          '((EXTENSION)  (PARALLEL)  (ORTHOGONAL))
          'R-O)
```

D. Results of the Pattern-Matching Procedure for These Examples

```
(FUNCTIONAL-MATCH 'TO-SCREW-1 'SCREWDRIVER-1)
TO-SCREW-1 = TYPE-OBJECT:  GRASPABLE
                           SCREWING  (SHAPE FLARED)  (LENGTH SHORT)
SCREWDRIVER-1 HANDLE --> (GRASPABLE)
              BLADE --> (SCREWING  (SHAPE FLARED)  (LENGTH MEDIUM))

MESSAGES:  NONE
```

The match is successful at the conceptual level without no requirement of parts.

```
CONCLUSION IS (1.0 0.33)

(FUNCTIONAL-MATCH 'TO-SCREW-2 'SCREWDRIVER-1)
TO-SCREW-2 = TYPE-OBJECT:
               TYPE-PART:  NAME X1
                           GRASPABLE
               TYPE-PART:  NAME X2
                           SCREWING  (SHAPE FLARED)  (LENGTH SHORT)
                CONNECTION X1 X2  (R-O EXTENSION)
SCREWDRIVER-1 HANDLE --> (GRASPABLE)
              BLADE --> (SCREWING  (SHAPE FLARED)  (LENGTH MEDIUM))

MESSAGES:
1 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X1)  (GRASPABLE))
       LOOK FOR A PART AMONG (HANDLE BLADE) TO AFFORD ((GRASPABLE))
```

The system looks first if the Handle is graspable.

```
2 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X2) (SCREWING
            (SHAPE FLARED) (LENGTH SHORT)))
     LOOK FOR A PART AMONG (HANDLE BLADE) TO AFFORD
            ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
2 <- ** HANDLE IS ALREADY AFFECTED --> SKIP IT
```

Even if the Handle can afford the required functionality it is skipped since the two functionalities have to be in two different parts.

```
3 -> LOOK FOR A PART AMONG (BLADE) TO AFFORD ((SCREWING
            (SHAPE FLARED) (LENGTH SHORT)))

4 -> A CONNECTION IS REQUIRED
     LOOK FOR A CONNECTION AMONG ((X2 BLADE) (X1 HANDLE))
          BETWEEN X1 AND X2 TO AFFORD ((R-O EXTENSION))

CONCLUSION IS (1.0 0.5)

(FUNCTIONAL-MATCH 'TO-SCREW-3 'SCREWDRIVER-1)
TO-SCREW-3 = TYPE-OBJECT: GRASPABLE
                          TYPE-PART:
                           NAME X2
                           SCREWING (SHAPE FLARED) (LENGTH SHORT)
                           CONNECTION X2 >X (R-O EXTENSION)
SCREWDRIVER-1 HANDLE --> (GRASPABLE)
              BLADE --> (SCREWING (SHAPE FLARED) (LENGTH MEDIUM))

MESSAGES:
1 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X2) SCREWING
            (SHAPE FLARED) (LENGTH SHORT)) (CONNECTION X2 >X
            (R-O EXTENSION)))
     LOOK FOR A PART AMONG (HANDLE BLADE) TO AFFORD
            ((SCREWING (SHAPE FLARED) (LENGTH SHORT))
            (CONNECTION X2 >X (R-O EXTENSION)))

2 -> A CONNECTION IS REQUIRED
     LOOK FOR A CONNECTION AMONG ((X2 BLADE))
            BETWEEN X2 AND >X TO AFFORD ((R-O EXTENSION))

3 -> LOOK FOR A PART AMONG (BLADE) TO AFFORD
           ((SCREWING (SHAPE FLARED) (LENGTH SHORT))
           (CONNECTION X2 >X (R-O EXTENSION)))

4 -> A CONNECTION IS REQUIRED
     LOOK FOR A CONNECTION AMONG ((X2 BLADE))
            BETWEEN X2 AND >X TO AFFORD ((R-O EXTENSION))
     LOOK FOR A CONNECTION AMONG ((X HANDLE) (X2 BLADE))
            BETWEEN X2 AND X TO AFFORD ((R-O EXTENSION))

CONCLUSION IS (1.0 0.33)

(FUNCTIONAL-MATCH 'TO-SCREW-1 'SCREWDRIVER-1-BIS)
TO-SCREW-1 = TYPE-OBJECT: GRASPABLE
                          SCREWING (SHAPE FLARED) (LENGTH SHORT)
```

```
SCREWDRIVER-1-BIS
    BLADE --> (GRASPABLE)
          --> (SCREWING (SHAPE FLARED) (LENGTH MEDIUM))
    HANDLE --> (GRASPABLE)

MESSAGES: NONE
```

The system found directly the Blade part to be good for the two required functionalities. CONCLUSION IS (1.0 0.33)

```
(FUNCTIONAL-MATCH 'TO-SCREW-2' 'SCREWDRIVER-1-BIS)
TO-SCREW-2 = TYPE-OBJECT:
             TYPE-PART: NAME X1
                        GRASPABLE
             TYPE-PART: NAME X2
                        SCREWING (SHAPE FLARED) (LENGTH SHORT)
             CONNECTION X1 X2 (R-O EXTENSION)
SCREWDRIVER-1-BIS
    BLADE --> (GRASPABLE)
          --> (SCREWING (SHAPE FLARED) (LENGTH MEDIUM))
    HANDLE --> (GRASPABLE)

MESSAGES:
1 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X1) (GRASPABLE))
     LOOK FOR A PART AMONG (BLADE HANDLE) TO AFFORD ((GRASPABLE))

2 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X2) (SCREWING
             (SHAPE FLARED) (LENGTH SHORT)))
     LOOK FOR A PART AMONG (BLADE HANDLE) TO AFFORD
             ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
2 <- **BLADE IS ALREADY AFFECTED --> SKIP IT

3 -> LOOK FOR A PART AMONG (HANDLE) TO AFFORD
             ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
3 <- BACKTRACKING
```

The Handle cannot afford the required functionality so the system will look into already located parts.

```
4 -> LOOK FOR A PART AMONG (BLADE) TO AFFORD
             ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
     BACKTRACKING NEEDED FOR ((TYPE-PART (NAME X1)
             (GRASPABLE))
```

An already located part is relocated but the previous functionality has to be re-examined.

```
5 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X1)
            (GRASPABLE))
5 -< **BLADE IS ALREADY AFFECTED --> SKIP IT

6 -> LOOK FOR A PART AMONG (HANDLE) TO AFFORD
            ((GRASPABLE))

7 -> A CONNECTION IS REQUIRED
     LOOK FOR A CONNECTION AMONG ((X1 HANDLE) (X2 BLADE))
            BETWEEN X1 AND X2 TO AFFORD ((R-O EXTENSION))

CONCLUSION IS (1.0 0.5)

(FUNCTIONAL-MATCH 'TO-SCREW-3 'SCREWDRIVER-1-BIS)
TO-SCREW-3 = TYPE-OBJECT: GRASPABLE
                          TYPE-PART:
                           NAME X2
                           SCREWING (SHAPE FLARED) (LENGTH SHORT)
                           CONNECTION X2 >X (R-O EXTENSION)
SCREWDRIVER-1-BIS
    BLADE --> (GRASPABLE)
          --> (SCREWING (SHAPE FLARED) (LENGTH MEDIUM))
    HANDLE --> (GRASPABLE)

MESSAGES:
1 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X2) (SCREWING
            (SHAPE FLARED) (LENGTH SHORT)) (CONNECTION X2 >X
            (R-O EXTENSION)))
     LOOK FOR A PART AMONG (BLADE HANDLE) TO AFFORD
            ((SCREWING (SHAPE FLARED) (LENGTH SHORT))
            (CONNECTION X2 >X (R-O EXTENSION)))

2 -> A CONNECTION IS REQUIRED
          LOOK FOR A CONNECTION AMONG ((X2 BLADE))
            BETWEEN X2 AND >X TO AFFORD ((R-O EXTENSION))

CONCLUSION IS (1.0 0.33)

(FUNCTIONAL-MATCH 'TO-SCREW-1 'SCREWDRIVER-2)
TO-SCREW-1 = TYPE-OBJECT: GRASPABLE
                          SCREWING (SHAPE FLARED) (LENGTH SHORT)
SCREWDRIVER-2 HANDLE --> (GRASPABLE)
              BLADE --> (SCREWING (SHAPE TORX) (LENGTH SHORT))

MESSAGES:
1 -> SET A PARTIAL RESULT
          ((PARTIAL-AFFORD (1.0 1.0) SCREWDRIVER-
          2 IN HANDLE FOR (GRASPABLE)))
```

The match failed globally but a functionality has been found at the conceptual level and will not be checked again at the structural level.

```
CONCLUSION IS FAIL --> TRIGGER INTERFACE MODULE
Q? (SCREWING (SHAPE FLARED) (LENGTH SHORT)) IN SCREWDRIVER-2
A = (0.6 0.3)
CONCLUSION IS (0.6 0.3)
```

```
(FUNCTIONAL-MATCH 'TO-SCREW-2 'SCREWDRIVER-2)
TO-SCREW-2 = TYPE-OBJECT:
             TYPE-PART: NAME X1
                        GRASPABLE
             TYPE-PART: NAME X2
                        SCREWING (SHAPE FLARED) (LENGTH SHORT)
             CONNECTION X1 X2 (R-O EXTENSION)
SCREWDRIVER-2 HANDLE --> (GRASPABLE)
              BLADE --> (SCREWING (SHAPE TORX) (LENGTH SHORT))

MESSAGES:
1 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X1)
          (GRASPABLE))
     LOOK FOR A PART AMONG (HANDLE BLADE) TO AFFORD
          ((GRASPABLE))

2 -> A PART IS REQUIRED FOR (TYPE-PART (NAME X2) (SCREWING
          (SHAPE FLARED) (LENGTH SHORT)))
     LOOK FOR A PART AMONG (HANDLE BLADE) TO AFFORD
          ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
2 <- **HANDLE IS ALREADY AFFECTED --> SKIP IT

3 -> LOOK FOR A PART AMONG (BLADE) TO AFFORD
          ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
3 <- BACKTRACKING

4 -> LOOK FOR A PART AMONG (HANDLE) TO AFFORD
          ((SCREWING (SHAPE FLARED) (LENGTH SHORT)))
     SET A PARTIAL RESULT
          ((PARTIAL-AFFORD (1.0 1.0) SCREWDRIVER-2 IN HANDLE
          FOR (TYPE-PART (NAME X1) (GRASPABLE))))

5 -> A CONNECTION IS REQUIRED
          LOOK FOR A CONNECTION AMONG ((X2 NIL) (X1 HANDLE))
            BETWEEN X1 AND X2 TO AFFORD ((R-O EXTENSION))

CONCLUSION IS FAIL --> TRIGGER INTERFACE MODULE
Q? (TYPE-PART (NAME X2) (SCREWING (SHAPE FLARED)
    (LENGTH SHORT))) IN (BLADE)
A = (0.6 0.3)
   WHERE? Blade
Q? CONNECTION HANDLE BLADE IS ((R-O EXTENSION))
A = (0.5 0.5)
CONCLUSION IS (0.5 0.3)
```

References

1. L.M. Vaina and M.C. Jaulent, "Object structure and action requirements: A compatibility model for functional recognition," *Intelligent System Journal*, **6**, 313–336 (1991).
2. R.A. Brooks, "Symbolic reasoning among 3-D models and 2-D images," *Artificial Intelligence*, **17**(1–3), 285–348 (1981).
3. A. Gimson, "Binocular shading and visual surface reconstruction," *Computer Vision Graphics and Image Processing*, **28(1)**, 19–43 (1984).
4. B. Bhanu, "Representation and shape matching of 3-D objects," *IEEE Pattern Anal. Mach. Intell.*, PAMI, **6(3)**, 340–350 (1984).

5. D.M. McKeown and J.F. Pane, "Alignment and connection of fragmented linear features in aerial imagery," in *Proceedings, IEEE Conf. Computer Vision and Pattern Recognition,* San Francisco, June 1985.
6. L.M. Vaina, "From shapes and movements to objects and actions: Design constraints on the representation," *Synthese* **54,** 3–36 (1983).
7. D'Arcy Thompson, *On Growth and Form,* Cambridge University Press, 1961.
8. L.A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems* **1,** 3–28 (1978).
9. M. Di Manzo, F. Ricci, A. Batistoni, and C. Terrari, "Using functional knowledge in computer vision," *IEEE Proceedings,* **74(7),** 1013–1025 (July 1986).
10. P. H. Winston, T. O. Binford, B. Katz, and M. Lowry, "Learning physical descriptions from functional definitions, examples and precedents," MIT Al Memo **679,** November 1982.
11. G. Ardoni, M. Di Manzo, F. Giunchiglia, and F. Ricci, "Building functional descriptions," *Proceeding ROVISEC 5,* Amsterdam, 1985.
12. L.M. Vaina, *Semiotics of With, Versus,* 17 Bompiani, Milano, 1978, pp. 96–112.
13. L.M. Vaina, "Towards a computational theory of semantic memory," in *Cognitive Constraints on Communication,* L. Vaina and J. Hintikka, Eds., 1984, pp. 97–113.
14. H. Blum, "Biological shape and visual science," *J. Theor. Biol.,* **38,** 205–287 (1973).
15. T. Binford, "Visual perception by computer," *IEEE Conference on Systems and Control,* Miami, 1971.
16. J. Koenderink and A. van Doorn, "The shape of smooth objects and the way contours end," *Perception,* **11,** 129–137 (1982).
17. D. Hoffman, and W. Richards, "Parts for recognition," *Cognition* **18,** 65–96 (1984).
18. B. Bennett and D. Hoffman, "Shape decompositions for visual recognition: The role of transversatility," in *Image Understanding,* W. Richards and S. Ullman, Eds., MIT Press, 1987, pp. 231–241.
19. L.M. Vaina and S.D. Zlateva, "A convexity-based method for extracting object from 3-D surfaces." *Proceedings SPIE,* 1991, to appear.
20. S.D. Zlateva and L.M. Vaina, "From object structure to object function," *Proceedings SPIE,* 1991, to appear.
21. J. Koenderink, "An internal representation for solid shape based on the topological properties of the apparent contour," in *Image Understanding,* W. Richards and S. Ullman, Eds., MIT Press, 1987, pp. 257–285.
22. T. Pavlidis, "Algorithms for shape analysis of contours and waveforms," *IEEE Trans. PAMI,* 2(4), 121–147 (1980).
23. D.H. Ballard and C.M. Brown, *Computer Vision,* Prentice-Hall, Englewood Cliffs, NJ, 1982.
24. D. Marr and H.K., Nishihara, "Representation and recognition of the spatial organization of three-dimensional shapes," *Proc. R. Soc. Lond.,* **B 200,** 269–294 (1978).
25. E. Rosch, C.G. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem, "Basic objects in natural categories," *Cognitive Psychology* **8,** 382–439 (1976).
26. A. Tversky, "Features of similarity," *Psychological Review,* **84,** 327–352 (1977).
27. D.L. Medin, W.D. Wattenmaker, and S.E. Hampson, "Family resemblance, conceptual cohesiveness and category construction," *Cognitive Psychology,* **19,** 242–279 (1987).
28. P.H. Winston, "Learning structural descriptions from examples," in *The Psychology of Computer Vision,* McGraw-Hill, New York, 1975.
29. D. Dubois and H. Prade, *Théorie des Possibilités: Applications à la Représentation des Connaissances en Informatique,* Masson, Paris, 1985.
30. M.C. Jaulent, "Un système souple pour identifier des objects géométriques plans décrits par un opérateur humain," Doctorat INP, Toulouse, France, 1986.
31. L.A. Zadeh, "Fuzzy sets," *Information and control,* **8,** 3–28