Abstract
Knowledge acquisition leads to the general problem of memory reorganization when a new information is to be integrated in the system. We show, in this paper, how a general algorithm can be used to integrate a new information into memory. A generalisation graph is developed by the algorithm using a comparison operation able to compare two descriptions of knowledge. We show how this operation can be performed according to different knowledge representations.

Keywords : Inheritance systems, generalization based memory, case-based reasoning, systèmes experts.

1. Introduction
The inheritance principle is used in various systems such as knowledge bases, programming languages, artificial intelligence and information systems. Knowledge bases are often based on a hierarchically structured memory allowing abstraction and generalization. These organizations save memory allocation and highlight similarities and differences between stored information. Entities can represent objects of a specific domain, events, scenes, planning situations, tasks,... It would be preferable to use a dynamic memory allocation method instead of leaving this task to the user. This method becomes a necessity when knowledge integration is a continuous process or when reasoning is based on memory (case-based memory). We propose, a general algorithm for addressing this problem that shows the dynamic propagation of memory, its structure and organization. The algorithm is independent of the type and the representation of knowledge to be integrated into memory.

Several algorithms have been developed to include the inheritance principle into a generalized hierarchy ([3], [9], [7]). In a cognitive approach, Schank [12] proposes a general theory of memory.
recall and learning principles. Inspired by this work, Lebowitz [8] and Kolodner have proposed several procedures that implement a memory based on the generalization principle. They have shown its usefulness in understanding, reasoning and planning process, leading to case-based reasoning ([7], [11]). This last approach is more and more considered as an alternative to expert systems architectures [5].

The algorithm presented in this article is based on three key elements: a general process of knowledge integration, a propagation principle that uses fundamental operation, and a similarity principle.

2. A general algorithm for dynamic creation of memory
2.1 Definitions
The basic elements of the algorithm are the concepts of situation, memory and comparison operation.

**Definition 1.** A *situation* can be an object, an event, a scene, a planning process, a procedure, a task or a part of a system. A situation can be described by a set, a predicate, a conceptual graph, a network of actors and objects ([4]) or any formalism of knowledge representation provided that we can define and apply a comparison operation between two situations. A situation may correspond to a new situation or to an existing situation in memory. If S is the name of a situation, s, the description or the content of S, we will note a situation by S/s.

**Definition 2.** The *memory* is a graph in which nodes represent situations and arcs inheritance links. Integrating a new situation into memory consists in propagating the situation through the graph and comparing the new situation with all the nodes.

**Definition 3.** The *comparison* operation COMPARE(input s,n; output c,s*,n*,i) gives the content c of a common situation C that exists between two situations S/s and N/n, extracts c from s (the result is then noted s*) and c from n (the result is noted n*), and returns in i, an operator, the result of the comparison.

COMPARE uses two main operations: COMMON (input s, n; output c) that aims at providing common elements between two situations, and EXTRACT (input c, a; output a*) that extracts these elements. The possible values of i are the following:

- i = "≠" if s cannot be compared with n (s ≠ n) then c = nil, s* = s, n* = n.
- i = ">" if s is more general than n ; s > n , c = s, s* = nil, n* = n - s. (n - s indicates that we extract s from n)
- i = "<" if s is more specific than n; s < n , c = n, s* = s - n, n* = nil.
- i = "=" if s is equivalent to n ; s = n , c = s, s* = nil, n* = nil.
- i = "‡" if s and n have common parts ; s ‡ n : c = s ‡ n, s* = s - c, n* = n - c

The following three cases are only examples (among several possible cases) which illustrate the representation of a situation and the compare operation.

**Case 1:** if the situation is described by a set of elements, then COMMON corresponds to the intersection operation and EXTRACT to the complement operation. Reconstructing a situation will be done using union of predecessors in the generalization graph.

**Case 2:** if the situation is described by predicates then the content of a situation is represented by a list of elements of the form <attribute,value> where attribute is an identifier and value a constant or a variable.

COMMON(s,n,c) can be defined as follows: for each element <A, V>, we search if there exists an element in n with the same attribute A, for instance an element <A, V1>; then we try to find a COMMON element V3 for V and V1. In case of success we add <A, V3> to the content c. The following rule applies: V et V1 can be constants or variables; if one is a variable then V3 must be a variable, if the two values are constants then V3 must be equal to this constant otherwise V has nothing in common with V1.

**Example:** If COMMON is activated with the following situations s and n :

s = [<Type, Micro-Wave>, <Price, 500>, <Manufacturer, [USA, Japan]>],

n = [<Type, Television>, <Price, 750>, <Manufacturer, [USA, Japan]>, <Lifetime, 10>]

the result obtained would be c = [<Type, t>, <Price, p>, <Manufacturer, [USA, Japan]>]

We see that c is more general than s and n.

**Example:** Suppose we have to propagate the situation S/s with the previous example.
EXTRACT \((c, n, n^*)\) and EXTRACT \((c, s, s^*)\) would give respectively \(n^*\) and \(s^*\), with:

\[
\begin{align*}
n^* &= [\langle \text{Type}, \text{Television} \rangle, \langle \text{Price}, 750 \rangle, \\
&\quad \langle \text{Lifetime}, 10 \rangle]. \\
s^* &= [\langle \text{Type}, \text{Micro-Wave} \rangle, \langle \text{Price}, 500 \rangle].
\end{align*}
\]

**Case 3**: if the situation is described by a conceptual graph CG, COMMON will correspond to the generalization of two CGs while EXTRACT will correspond to the contraction; reconstructing a situation will be done by joining the two CGs (Kabbaj [92c]).

### 2.2 Fundamental principles

- **Principle of propagation**

  This principle precises how a situation is propagated into memory. Figure 1 shows a part of the memory at a given time. Suppose that the integration process has propagated a new situation \(S/s\) until node \(B/b\). We wish to compare \(s\) with \(b\).

  If \(s < b\) (the content of \(S\) is more specific than the content of \(B\)) then we extract \(b\) from \(s\) and we propagate \(S/s-b\) in parallel through the children of \(B\) (figure 1).

  Now, if \(S/(s-b) < E/e\), we apply the same principle, extract \(e\) from \(s-b\) and propagate the result to the sons of \(E\). However we must note the problem of "partial inheritance": the propagation of \(S\) via \(E\) is continued so we assume that \(S\) inherits from \(B\) (which is true) but also from the other fathers of \(E\) (which is not verified). We will see how the algorithm deals with this problem of partial inheritance.

- **Principle of similarity**

  If, after propagating \(S\) via \(B\) and comparing \(S\) with \(E\), we find that they have nothing in common then the algorithm does not propagate \(S\) via \(E\); yet we could miss the detection of similarities between \(S\) and descendants of \(E\) unless \(S\) be propagated to these nodes through other paths using parallel propagation. In fact, this partial process of similarity seems to reflect a specific behaviour of human memory: after acquired a situation \(S\), we cannot make all the connections with all the nodes in memory but as the memory changes perpetually, it can arrive that following a new integration of the situation \(S\), we could identify similarities that without any apparent similarity.

![Figure 1: A part of the memory](image)

How many times we exclaimed: "Oh yes, it's true, I did not notice before!" when we link two situations previously known but that were hidden before.

### 2.3 Fundamental characteristics of the algorithm

We suppose that the memory has a root node from which the integration process can be initialized. When a situation \(S\) is integrated, the content of the memory can be empty. **Integrating** the situation \(S/s\) consists in creating a node in memory, with \(s\) as initial content, and in **propagating** \(S/s\) via the root node. In general, the content of the node \(S\) will be updated and linked to others nodes either as father or son. During this propagation, other arcs and links can be created or destroyed. Finally, after the integration of a situation, it may be necessary to optimize the memory by eliminating redundant nodes and arcs.

The problem consists in integrating \(S/s\) via a **current node** \(N/n\), \(S/s\) resulting from the node \(M/m\). Two principal cases are to be considered...
according to wether N has a single father or multiple fathers.

**Case 1 : the current node has a single father**

The initial state is described on figure 2.a. Five transformations can occur according to the result of the comparison of S/s and N/n.

- If s cannot be compared with n (s ≠ n), the propagation of S via N is cancelled and no change is made in memory.

The others cases are described in figure 2 (b, c, d and e):
- If s is more general than n (s > n), we eliminate the link MN and create links MS and SN; we extract s from n (which gives n-s in N) and we update the content of node S : S/(s1 + s).
- If s is equivalent to n (s = n), we replace the node N by updated S, i.e. N/n is replaced by S/(s1 + s), and we maintain the links attached to the two nodes.
- If s is more specific than n (s < n), we propagate S/(s - n) to the sons of N. If N has no son or has sons that cannot be compared to S then link NS is created with a new content for S : S/(s1 + (s - n)).
- If s and n have common elements (s ∩ n), the node C/c is created with c = s ∩ n, the link MN is deleted and links MC, CN and CS are created; we update the contents of N : N/(n - c) and S : S/s2, with s2 = s1 + (s - c).

**Case 2 : the current node has multiple fathers**

Now, we consider the case in which N has several fathers, leading to the problems of partial inheritance and multiple integration mentioned above. We will examine the different cases: integration of the first version sp of S arriving in N/n via M/m and integration of the other versions of S arriving in N through other nodes.

- If sp ≠ n then there is no change ; C = nil (i.e. there is nothing in common between sp and n) and so for the other versions sj of S.
- If sp > n then the transformation is similar to figure 2.b : the situation described by a is transformed as follows : we replace the link MN by MS and SN, extract sp from n and update the content of S (figure 3a). For each new version sj, passing through Mj father of N, we must eliminate the link MjN and create the link MjS (figure 3b).
- If sp = n then the transformation is the same as in the previous case (i.e. sp > n) except that the content of N becomes nil. There is no merge of the two nodes S and N because S does not necessarily inherit from the other fathers of N. The new versions are processed as in the previous case, eliminating MjN and creating MjS.

**Figure 2 : Case with a single father for N**
- If $sp < n$ then the transformation is different from what is described in figure 2.d. In fact, the situation illustrated by X.a is transformed as follows: a node G/n is created, N/n is replaced by N/nil, MN is replaced by the links MG and GN, and S/(sp - n) is propagated to the sons of N with G as father (figure 4.a). Creating the node G/n and precising the father G, we avoid partial inheritance. If N has no son or sons that cannot be compared with S, then the link GS is created and the content of S is updated with S/(s1 $\uparrow$ (sp - n)).

Consider now the arrivals of the other versions of S: if a new version sj passing through Mj (a father of N) arrives we eliminate the link MjN and create the link MjG (figure 4.b). The content of S is updated. If, for a new version sj of S we have sj = n, then S and G must be merged.

- If $sp \uparrow n$ then the transformation is similar to figure 2.e. When a new version sj arrives, passing through a father Mj of N, the link MjN is replaced by MjC, and again, if a new version sj of S contains only c or if the cumulative intersection of the versions contain only c, then we merge S with C. A more detailed justification of the transformations mentioned above is given in Kabbaj [92a].

3. Applications

Several applications have been developed to illustrate the useful purpose of a memory based on the generalisation such as: classification, text understanding, planning and reasoning. We plan to experiment some of these applications with our model. The model is a main component of a knowledge representation system based on the conceptual graph theory which can be used in a variety of applications in AI, intelligent tutoring systems, intelligent databases and information systems. Specific applications in intelligent tutoring systems are under way.

A major application of this model can be found in the SMAM project which consists in developing a multi-agents system in which each agent incorporates a specific behavior with human characteristics (knowledge, belief, intention, feeling, emotion, character,...). Agents can communicate through images or pseudo-natural language. Each agent has its own memory that is used by different cognitive process. Our intention is to model the memory behavior of an agent in a natural context close to the human.

Another application of this approach aims at determining a memory for a multimedia tutoring
system, the situations corresponding to texts and images.

4. Conclusion

Systems based on the inheritance principle are used in object oriented programming, object databases, object oriented analysis methods and knowledge representation in AI. They use, as a knowledge base, a hierarchy of abstraction in which each node inherits from its predecessors and two nodes share information contained in their common predecessors. Several researchers have proposed algorithms for a dynamic creation of the hierarchy, information retrieval and hierarchy-based reasoning.

The model of memory proposed in this article can be considered as a fundamental element of a "natural" memory based on the generalisation principle. The memory is natural in the sense that all the similarities between situations may not always be detected, as in human memory. We plan to use this model as a support for intelligent tutoring systems, in learning environment and reasoning analysis. A more complete description of the model is given in [4].

References