An architecture for selective forgetting

Jérôme Euzenat(1), Libero Maesano(2)

(1) Sherpa project, Laboratoire ARTEMIS/IMAG, BP53X, F-38041 GRENOBLE
(2) CEDIAG/Bull, 68 route de Versailles, F-78430 LOUVECIENNES

ABSTRACT

Some knowledge based systems will have to deal with increasing amount of knowledge. In order to avoid memory overflow, it is necessary to clean memory of useless data. Here is a first step toward an intelligent automatic forgetting scheme. The problem of the close relation between forgetting and inferring is addressed, and a general solution is proposed. It is implemented as invalidation operators for reasoning maintenance system dependency graphs. This results in a general architecture for selective forgetting which is presented in the framework of the Sachem system.

INTRODUCTION

The generalized computerization of the organizations leads to a worrying amount of stored data. The problem of forgetting is an old one: it is simply the ability to avoid memory overflow by freeing it. Moreover, freeing memory does not only care for memory overflow but also ensures good performances of data retrieving procedures. So, this problem is critical for data management. Computer systems are yet able to exhibit forgetting skills. In several programming languages, the programmer can use some instructions in order to free the occupied memory. In more evolved languages, a garbage collection mechanism is able to find out unworkable data that can be discarded from memory. Concerning secondary storage, every operating system provides "remove file" commands and every data base management system "remove record" ones. It is possible to go ahead in the mechanization of forgetting and to put forth intelligent forgetting tools.

It is possible to consider a simple forgetting scheme (with or without archive of forgotten data) for the future database systems. It can be a very basic system (banks must keep information on transactions for ten years) or a more sophisticated and intelligent one.

Intelligent forgetting capabilities will be studied in a knowledge based system context for reasons of declarativity of the knowledge expressed and of availability of exploitation tools. The relations between forgetting and inferring will be especially accounted for: rather than being a problem, they allow building a smarter forgetting scheme taking advantage of inferences. It will be demonstrated through a real world application in which these ideas were experienced. This study will not distinguish between primary and secondary storage memory because it is concerned with very large applications that must reside in databases.

in: Luc Steels, Barbara Smith (eds.), AISB91 (proceedings of the 8th SSAISB conference on artificial intelligence and simulation of behaviour), Springer-Verlag, London (GB), 1991

Forgetting will be considered together with knowledge base systems, and so, will meet the problem of the relations between inferring and forgetting. These relations will be dealt with through two attitudes: consolidation and abstraction.

Reasoning maintenance systems will be introduced in order to demonstrate the implementation of these two attitudes. This will lead to the presentation of a complete architecture for selective forgetting, together with a proposal enabling to determine what to forget. This is presented through its application to a knowledge based system which monitors a blast furnace process: Sachem.

It is worth noting that this work concerns artificial systems, and so, is not preoccupied by psychological plausibility. Nevertheless, it obviously can and should be related with psychological studies and psychological preoccupation in computer science. This will be addressed in the last section.

FORGETTING AND INFERRING

When considering knowledge based systems, forgetting some datum must be considered in connection with every other part of the base it is connected with. The problem to be faced is that of the attitude toward the relation after forgetting: discarding relations or discarding the related item (and so on...). Moreover, the relations can be of two sorts:

- *Explicit relations* can be found as references from an object to another. Knowledge representations and, among them, object based knowledge representations allow for a wide range of relations to be introduced in the base and a rich semantics associated with each kind of relation (Kim & al., 1989; Escamilla & Jean, 1990).
- *Implicit relations* need computation in order to be established. This is the case for the relations from classes to instances through the inheritance mechanism but also for every deductive inference mechanism.

Only the relations between forgetting and inferring are discussed here. This will lead to a safe and generic solution for this relation. A deductive relationship between datum to forget and other data is twofold: a backward relation with other data that enabled to infer it and a forward relation with data it allowed to infer. Both sides of the relation are discussed below.

Backward references

Forgetting is not of logical concern. Logic tells us what interpretation can be given to a set of facts while forgetting changes the set of facts considered. So, as for other problems (reasoning maintenance for example), the justification of forgetting is to be found in reasoning rather than logic. However, in order to insure a sound behavior of the system, we want it to have a semantics that warrants that every item which can be inferred is indeed. For that purpose, an inference system is required to infer every formula that is inferable from a set of facts. So, the problem to be faced is that of the meaning of forgetting inferred knowledge: the facts introduced into the database by the inference system. In fact, after forgetting an item, it should be legitimate for the reasoner to infer it again, perhaps *ad infinitum*. Moreover, forgetting the conclusions while the premises remain is not appropriate: usually, the inference process is oriented toward inferring gradually more relevant data, so, it is better to preserve relevant data while forgetting rough and less relevant data.

Our choice, consists in forgetting only what is called initial knowledge (i.e. knowledge that the logical interpretation of the base considers as axioms). This is the price to be paid for insuring a logical behavior of the system.

This accords with the observation that only more abstract data is inferred (through true abstraction or aggregation): it is a suitable approach to forget initial knowledge rather than more abstract knowledge. In fact, if the system is dedicated to infer abstract knowledge (and this is mainly how the Sachem system behaves), it seems curious to forget abstract knowledge while keeping at hand the initial knowledge that generated it.

Forward references

On the other hand, if one wants to forget initial data, (s)he may want to forget what that data allowed to infer. In fact, there are two possible attitudes:

- Forgetting data while leaving everything else unaltered, which will be called *consolidation*. This can be used in order to forget initial data while preserving relevant consequences of those data.
- Forgetting data while leaving everything else as if the forgotten datum was in a particular state (usually valid or invalid), which will be called *abstraction*.

It is noteworthy that the latter does not rely on a current state while the former does. But those two possibilities are not independent: the former can be expressed by the latter with the state conditioned by the actual state of the datum, the latter can be expressed by changing the state of the datum before applying consolidation. Both attitudes allow for logical interpretation: the forgotten data, which was axioms are not any more, and so their consequences are invalid or become axioms in turn. However, the presented attitudes have been stated in a very abstract fashion: even the forward references are not mentioned. It must be instantiated in particular deductive systems.

These forgetting primitives are very interesting and must be provided, but their implementation is not straightforward because axioms and consequences are not usually explicitly related. The architecture presented below deals with such a problem, but first, the implementation of the operators is introduced.

REASONING MAINTENANCE SYSTEMS

Reason maintenance systems (RMS) are aimed at managing a knowledge base considering different kinds of reasoning. Such a system is connected to a reasoner (or problem solver) which communicates every inference made. The RMS has in charge the maintenance of the reasoner's current belief base. Reasoning maintenance systems' dependency graph will form the basis on which the operators can be implemented. So reasoning maintenance systems are first exposed before showing how to implement forgetting operators.

Dependency graph

RMSes record each inference in a *justification* that relates *nodes* representing propositional formulas plus a special atom ($\perp$) representing contradiction. The system accepts non monotonic inferences so the justifications have an appropriate structure: a justification ($<\{i_1,\ldots i_n\}\ \{o_1,\ldots o_m\}>$: c) is made of an IN-list ($\{i_1,\ldots i_n\}$) and an OUT-list ($\{o_1,\ldots o_m\}$). Such a justification is said to be valid if and only if all the nodes in the IN-list are known to hold while those in the OUT-list are not; a node, in turn, is known to hold if and only if it is the consequent (c) of a valid justification. The recursion of the definition is stopped by nodes without justification and by axioms that are nodes with a justification containing empty IN- and OUT-lists.

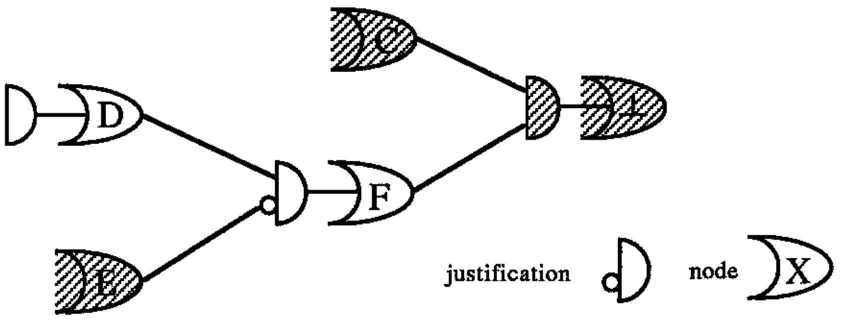

Fig. 1. A dependency graph is here represented as a boolean circuit where or-gates are nodes and and-gates are justifications where the nodes in the IN-list come directly while nodes in the OUT-list come through a not-gate. Nodes that have a justification whose IN- and OUT-lists are empty (e.g. D) represent true formulas because they do not need to be inferred. White nodes and justifications are considered valid while hatched ones are invalid. Of course, the value propagation satisfies the rules implied by the circuit components. So, the formulas in the base are insured to have a valid justification (i.e. corresponding to a valid inference).

TMS and ATMS

Jon Doyle's TMS (for "truth maintenance system"; Doyle, 1979) proceeds by labelling the nodes of the graph with IN and OUT tags which reflect whether they are known to hold or not. A labelling respecting the constraints stated above is an admissible labelling and a labelling which labels the node $\perp$ OUT is a consistent labelling. The TMS algorithm finds a (weakly) founded labelling, i.e. a consistent admissible labelling which relies on no circular argument. The main work of the TMS occurs when it receives a new justification. It then has to integrate the justification in the graph and, if this changes the validity of the formula, it must propagate this validity: all the nodes that could be IN-ed because of the justified node and all those which could be OUT-ed are examined and updated. If an inconsistency occurs following the addition of a justification, the system backtracks on the justifications in order to invalidate a hypothesis — a formula inferred non monotonically — which supports the inconsistency.

Johan De Kleer's assumption-based TMS (Martins & Shapiro 1988; De Kleer, 1986) is rather different. This system considers only monotonic inferences (with only IN-list: $<\{i_1,\ldots i_n\}>$: c), but it deals with several contexts at a time. It considers initial formulas called hypotheses; so, the user can generate and test hypotheses with great efficiency. A set of hypotheses is called an environment and the set of all the environments constitutes a complete lattice structured by the "includes" relation (cf. Fig. 2). Instead of labelling absolutely a node (with IN or OUT tags), each node has a label consisting of the set of environments under which it is known to hold. An environment is consistent if $\perp$ is not known to hold in it and the computed labels are minimal in the sense that they do not contain comparable environments. After each inference, the system computes the set of environments that support the inference, inserts it in the label of the inferred node and propagates it through the graph. Then, in order to know if a formula is valid, it compares the current hypothesis set with the label of the node.

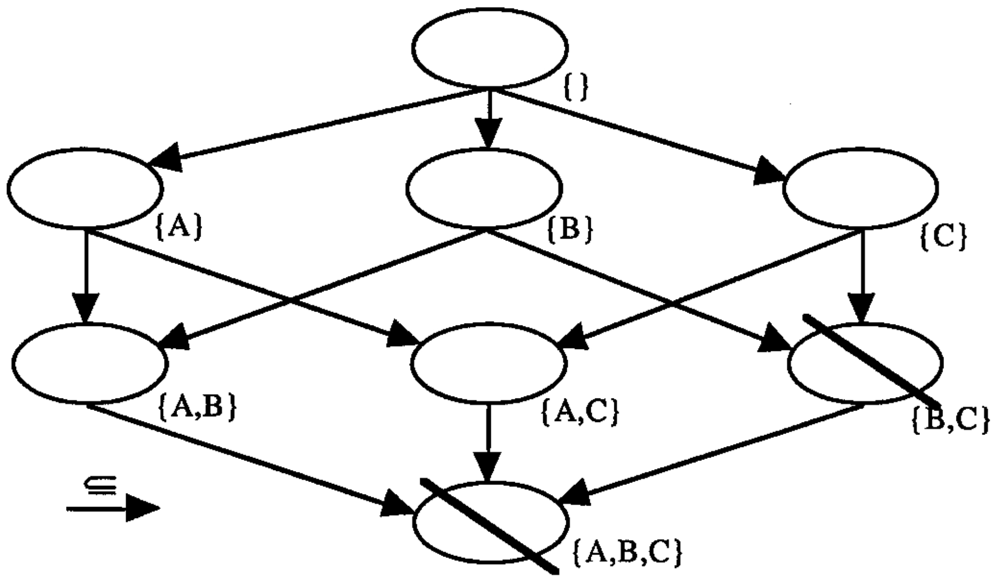

Fig. 2. The environment lattice constructed with the hypotheses A, B and C in which the environment {B, C} is known as inconsistent.

As a summary, the TMS handles non monotonic inferences and is able to maintain the set of deduced formulas with regard to an axiom set. The axiom deletion, while not explicitly described by Jon Doyle, is trivial to implement. The ATMS, for its part, cannot accept non monotonic inferences, but is able to consider several contexts simultaneously.

Specialized graph operators

Reasoning maintenance systems record every inference, but, independently of TMSes, the dependency graph constitutes a picture of the reasoning process and every manipulation of this graph can be seen as meta-reasoning. It is then possible, for other utilities, to take advantage of the graph. This is the case for:

- explanation generation,
- automatic inconsistency recovery (the task of backtracking),
- forgetting.

The forgetting primitives will be implemented, at a low level, as dependency graph procedures called *inferential* and *influential invalidation* that are run against a TMS node and a justification in which it appears. The former enables forgetting the justification, and, the latter forgets the effects of the node on the justification. Influential invalidation consists in replacing the justification in which the node appears (in IN- or OUT-list) by another in which the node no longer appears. Inferential invalidation suppresses the justification in which the node appears. Then, the reasoning maintenance propagation process is run in order to account for the change in the rest of the graph.

This last operation is not native in the ATMS (because it is a monotonic system), but if the forgotten fact is a hypothesis the result of the propagation will consist in a simple operation on the labels. Two logical concepts, called *universal* and *existential abstraction*, have been established at the Bull research center (Coudert & Madre, 1990) and are the simple quantification of boolean formulas by some variable (which ranges over 0 and 1). If applied to the labels, considered as formulas in disjunctive form, these operators formalize, in sentential calculus, the two possible operations of forgetting a fact with or without taking care of its consequences. So, the propagation phase of these operators is implemented by applying the relevant operator with the hypothesis to each consequence of the hypothesis.

Example

If a node X has for label {{A,B},{B,C},{A,D}} which can be expressed in sentential calculus by $(A \wedge B) \vee (B \wedge C) \vee (A \wedge D)$ then

$\forall B\ (A \wedge B) \vee (B \wedge C) \vee (A \wedge D)$ (universal abstraction)

$= [(A \wedge D)] \wedge [(A) \vee (C) \vee (A \wedge D)]$

$= (A \wedge D)$: {{A,D}} is its OUT-abstraction after B's forgetting and

$\exists B\ (A \wedge B) \vee (B \wedge C) \vee (A \wedge D)$ (existential abstraction)

$= [(A \wedge D)] \vee [(A) \vee (C) \vee (A \wedge D)]$

$= (A \vee C)$: {{A},{C}} is its IN-abstraction after B's forgetting.

Invalidation operators are far away from the consolidation and abstraction that were introduced previously. Here is sketched their implementation in terms of invalidation. The consolidation is not explained provided that it is straightforwardly adapted from abstraction.

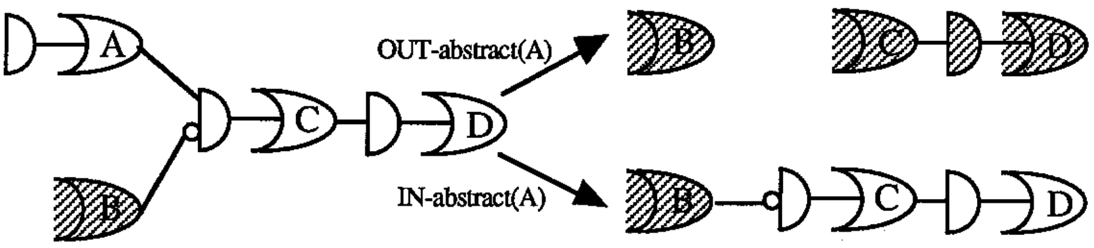

Fig. 3. Effect of abstraction on the dependency graph through invalidation operators.

The IN- or OUT-abstraction operators first check for the validity of the operation: the data must be an axiom (or an hypothesis), i.e. it cannot have been inferred. Then, the invalidation operators are called against the justifications in which the node appear. For IN-abstraction, the influential invalidation is called against the justification in which the node is in the IN-list and inferential invalidation against those in which it appears in the OUT-list. For OUT-abstraction, it is exactly the opposite. At last, the node can be suppressed from the node base.

Nevertheless, the invalidation operators are very powerless since they only allow the forgetting of one node at a time. So, two additional procedures are provided that are called recursive invalidation. They recursively apply invalidation to the consequences of the initial fact, provided that the operation is also valid (the consequence has become in turn an initial fact). It is noteworthy that these new operations necessitate only one propagation because the additional operations respect the current logical interpretation.

SACHEM FORGETTING PROCESS

This approach to forgetting stems from a large real-time knowledge-based system in which data comes from 800 sensors. This system diagnoses failures in the process of a blast furnace. It must infer from rough data (coming from the sensors) higher level representation of the process. From this representation, the system is able to make several diagnostics. They are represented by assumptions of failure. The system will then observe (or focus) on precise data in order to confirm, infirm or invalidate these hypotheses. Since it works in real time, it must also decide how to act in order to recover from the failure. These decisions take into account the irreversibility and the range of such actions and the plausibility of the hypotheses that support them.

The application is written in the object oriented language KOOL (Lacroix, 1989) enhanced with reasoning maintenance capabilities (Euzenat, 1989a). It uses objects for representing data and hypotheses and rules for reasoning. The tasks of the system and their schedule is described in Fig. 4.

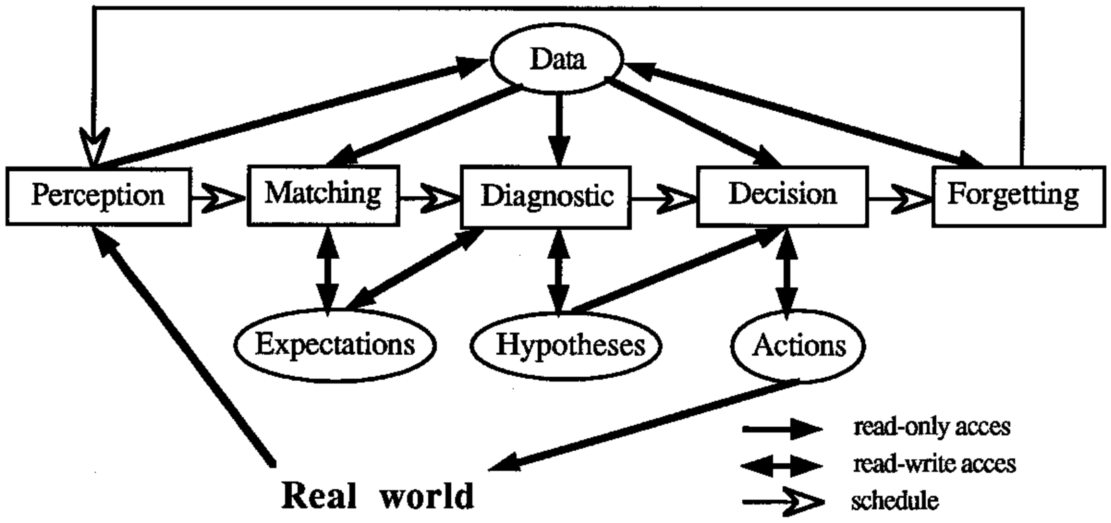

Fig. 4. Sachem general process.

The system receives large amount of data, and also constructs large amounts. This leads to important storage requirements and forces forgetting a part of this data in order to not overflow memory. The real-time constraint rules out the possibility of stopping the system in order to clean memory. But, it is loose enough so it is possible to run a system that will decide what to forget and thus free memory.

"What to forget?" versus "What to remember?"

There are two ways of achieving forgetting: the first one provides forgetting operators to the user who has in charge the explicit liberation of space, the second one, close to the garbage collector approach, uses a supervisor in order to decide what and when to forget. Of course, the supervisor can be driven by specific rules. We decided to use the second approach as shown by Sachem's main cycle.

When required to free memory, computer systems are usually able to do it, if they are told which places or which items to free. So, the whole problem of automating forgetting is to tell the computer what to forget. The problem of finding "what to forget" has been pointed out, but one can address the dual problem of "what to remember". Both answers are suitable in order to determine what to forget. The latter has been chosen for Sachem. It consists of establishing a set of relevant data (called the focus of attention) and forgetting everything which is not related to that set of data.

However, the problem of circumscribing that focus of attention is not solved. There is no all-purpose method. The solutions can be some automatic methods such as declaring initial data older than some date to be irrelevant, or heuristic methods, using inference upon the current state of the process in order to decide what to remember.

In Sachem, both methods are used. Validity duration of data and hypotheses leads to discarding them after some time or not discarding them before another time. The focus of attention is also used; it is based on a numeric comparison of the concurrent diagnostic hypothesis. They are assigned a plausibility factor which can change during their lifetime.

There is a bound under which an hypothesis is considered as no relevant any more: such an hypothesis will not appear any more in the focus of attention, and equally for the initial data that led to propose this hypothesis. Another approach, more domain dependent, is under investigation: it consists of imitating the way an expert manages his own focus of attention and necessitate special expert knowledge.

However, this problem is still open and a lot of different solutions can be considered. Now, will be presented the architecture which allows a close integration of the mechanisms provided so far. It is a general architecture and protocol that enable forgetting in knowledge based systems.

A knowledge-based system architecture for forgetting

The software architecture of Sachem is made of three levels (Euzenat, 1989b). The upper level is the fact base on which the inference methods can be run: this is the KOOL object level. The medium level is a reasoning maintenance dependency graph, representing each formula manipulated by the inference methods as a node and relating them by justifications which represent inferences. The lower level is made of the data structures of the implementation programming language (Lisp in this case). The novelty of the architecture stems from the way it is used for forgetting.

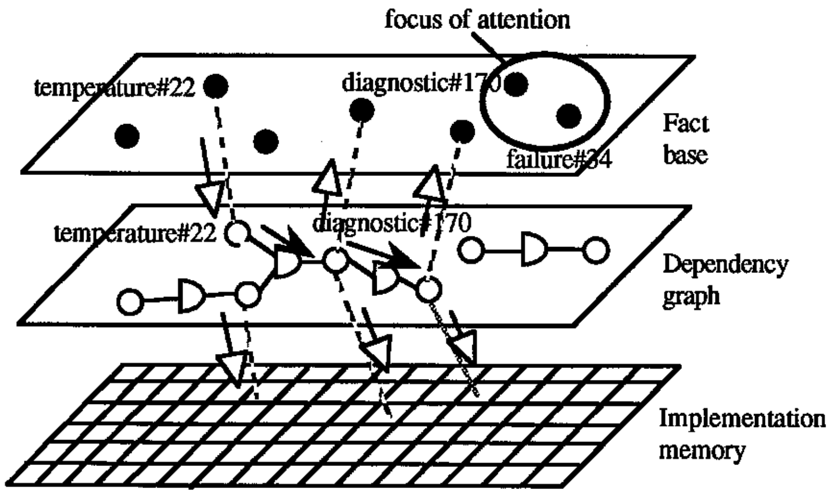

Fig. 5. The three level architecture of Sachem. The arrows represent the propagation of forgetting: from high level representations to lower levels (white arrows), from raw to more abstract data (black arrows).

This architecture leads to a simple protocol for forgetting. The forgetting process, as said above, is a high level mechanism, so it begins at the higher level by determining the focus of attention. Every item that has to be forgotten is then discarded from the fact base and a query for discarding it at the medium level is emitted. At the reasoning maintenance level, the invalidation operators are provided which discard recursively the concerned facts and their references from other structures. At the implementation language level, the garbage collector is able to collect the free memory. Of course, the reasoning maintenance level tells the knowledge base level to discard the consequences it has identified.

FORGETTING AS AN OPTIMIZATION PROCESS

Up to now, our preoccupations only relate to the computer implementation of reasoning systems. But this work can be compared with more general ideas and theories such as those on human cognitive forgetting.

Psychological insight

Psychology works distinguish between short term and long term memory (LTM). The analogy with the work presented here can only be done with LTM. There are two main models of retrieval in memory that account for forgetting (Tiberghein, 1987):

- Interference theory in which the item to be found is inhibited by a reorganization of memory (due to new information arrival).
- Context theory in which the storage context is so different to the retrieval context that the retrieval procedures are unable to find the item.

Generally, psychological models of forgetting, more than storage failure, emphasize retrieval failure. Storage failure, as opposed to retrieval failure, is non reversible and thus more difficult to test safely. But, such models show that forgetting reveals the optimization of the retrieval process: forgetting is the price to pay for retrieving quickly, almost every time. The view of forgetting presented here, while it differs from the ones above, is also an optimization, but a storage optimization. It also has its shortcomings: sometimes, items that had been known by the systems are no longer remembered. While the current psychological theories account for cognitive economy of processing, our forgetting proposal concerns cognitive economy of storage (Lenat & al., 1979).

However, studies in psychology should be interesting for artificial intelligence if they can show how workable part of memory is circumscribed: this could bring new ideas about how to forget while insuring coherency of considered data.

Abstraction

There is previous work in artificial intelligence which uses psychological models in order to build memory systems. For example, Roger Schank (Schank, 1982) considers that forgetting of individual data is the consequence of its integration in a more general scheme (in his case it was more general instances of scripts). This is also true in the psychological context theory of forgetting that observe that if something is learned under a wide range of contexts, the context does not matter anymore: the learned item is abstracted from the context.

The abstraction concept can take this into account. Moreover, that concept can be found in each context where a forgetting tool can be useful:

- In reasoning, it is possible to forget initial data which allowed to infer more synthetic and pertinent data. This approach can be related with the actual work on deepness level of reasoning (Bonté & al., 1988) which can help to determine what can/must be forgotten or not.
- In a scientific discovery perspective, every fact that is singular and contradictory with the current theory is very active in memory because it has to be explained. When explained with the help of a new theory, the abstraction, that the new theory represents, remains while the singular fact is forgotten because it is not singular any more (it is just an instance of the abstraction).

- In computer vision, pixels that constitute the input of the system are forgotten for the benefit of edges which are themselves forgotten in objects' advantage...
- In symbolic machine learning, when minimal and maximal representations of the concept to learn are acquired, examples and counter-examples from which they are built can be forgotten (this is the case of the restaurant-script example of Schank).

All of these categories can be taken into account in order to help choosing the items to forget. For example, Sachem's reasoning goes through several abstraction levels. The possibility of several different forgetting politics tied to the different levels are currently under study.

Of course, all these forgetting actions must only be processed when the infered/extracted/learned abstractions are strong enough. The forgetting action disables the explanation of the abstraction processes. So the choice of the items to be forgotten must be very careful.

As a conclusion, several concepts constructed for psychological purposes can be confronted with the forgetting ability described here. While not inspired from them, the contexts of ATMS share the same purpose with psychological contexts theory: reducing the search space. Schank abstraction is the same as that which is used here. Moreover, the understanding of forgetting schemes as an optimization can open some interesting artificial intelligence perspectives such as using encoding and decoding schemes inspired of those provided by psychological research (this is, of course, yet true for the computationalistic part of psychology, e.g. (Anderson, 1976)).

CONCLUSION

An "attitude" toward the relations between forgetting and inferring in knowledge based systems has been presented. Through abstraction and consolidation operators, forgetting can be safely dealt with. Moreover, it is possible, with the help of a reasoning maintenance system, to implement those operators. This leads to a proposal for a generic architecture which is suited to the purpose of forgetting and which has been implemented for the Sachem application prototype.

The forgetting operators meet the meta-reasoning activity which is the work of the reasoning maintenance system. At first sight, the action of forgetting can be seen as antithetic with reasoning maintenance systems which recall every inference produced in view to support defeasible reasoning. Moreover, reasoning maintenance systems seemed to burden large applications with too much data. But, at last, they turned out to be of major relevance when discarding an item must be propagated to its consequences.

The architecture presented here is a very generic one. Currently, many systems based on the object principle are developed, including object-oriented databases. There is discussions in order to know if these systems will be provided with garbage collectors or explicit freeing operators. Our claim is that garbage collectors are better, but not as they are currently understood. Classic garbage collectors are too low level tools for objects. In the most part of object applications, each object is connected with the entire base and has a lot of connections. So, it is not suited to ground the destruction operation on the lack of connections that which will not be achieved often. It is necessary to define the criteria that made an object "garbageable". For that purpose, a generic system that has in charge to apply the criteria to the objects in order to determine which object must be forgotten is proposed. This is done, in Sachem, through the determination of the focus of attention.

Farther than deductive implicit relations, explicit relations such as those that are definable in object based knowledge representations are actually under investigation. The taxonomy of these relations will control the spreading effect of forgetting (e.g. relation of composition or class-membership should lead to specialized forgetting policies).

Our architecture is designed for forgetting in a reasoning system. As mentioned above, forgetting is useful in several contexts. We think that this architecture can be generalized. In particular, reasoning maintenance systems can be applied, for several reasons, to learning systems.

Forgetting is a major concern as far as data and inference memorization is used in large applications. The starting research on that topic, especially if the efforts take into account the yet available and foregoing results of psychology and logic, should lead to very useful methodological tools.

ACKNOWLEDGEMENT

This work was initiated when the first author was with CEDIAG. The authors are grateful to Martin Strecker who pointed out some shortcomings in the first formulation of these ideas. Of course, remaining ones are under authors responsibility.

REFERENCES

Anderson J. R. (1976), Language, memory and thought, Lawrence Erlbaum associates, Hillsdale

Bonté E., Castaing J., Grandemange P., Grumbach S., Kayser D., Lévy F. (1988), Description succincte d'un raisonneur à profondeur variable, proc. 8ièmes journées internationales sur les systèmes experts et leurs applications, EC2, Avignon, 117-132

Coudert O., Madre J.-C. (1990), Logic over finite domain of interpretation: proof and resolution procedures, Bull research center, Louveciennes (Research report)

De Kleer J. (1986), An assumption-based TMS, *Artificial intelligence* 28(2):127-162

Doyle J. (1979), A truth maintenance system, *Artificial intelligence* 12(3):231-272

Escamilla J., Jean P. (1990), Relationships in an object knowledge representation model, proc. IEEE Conference on tools for artificial intelligence, Herndon, 632-638

Euzenat J. (1989a), Connexion KOOL/RMS, spécifications, CEDIAG/Bull, Louveciennes (Internal report Sachem JE004)

Euzenat J. (1989b), Démonstrateur B: architecture et fonctionnement, CEDIAG/Bull, Louveciennes (Internal report Sachem JE007)

Kim W., Bertino E., Garza J. (1989), Composite objects revisited, *SIGMOD records* 18(2):337-347

Lacroix V. (1989), KOOL: a reflexive representation language, proc. TOOLS 89, Paris, 309-321

Lenat D., Hayes-Roth F., Klahr P. (1979), Cognitive economy, Stanford university, Stanford (Research report HPP-79-15)

Martins J., Shapiro S. (1988), A model for belief revision, *Artificial intelligence* 35(1):25-79

Schank R. (1982), Dynamic memory, a theory of reminding and learning in computers and people, Cambridge university press, Cambridge

Tiberghein G. (1987), Introduction aux concepts contemporains dans l'étude de la mémoire chez l'homme, in: Martial Van Der Linden, Raymond Bruyer (eds.), proc. conference sur la neuropsychologie de la mémoire humaine, Société de Neuropsychologie de Langue Française, 2-31