11

An Approach to the Frame Problem, and its Implementation

E. Sandewall

Computer Sciences Department
Uppsala University

Abstract

The frame problem in representing natural-language information is discussed. It is argued that the problem is not restricted to problem-solving-type situations, in which it has mostly been studied so far, but also has a broader significance. A new solution to the frame problem, which arose within a larger system for representing natural-language information, is described. The basic idea is to extend the predicate calculus notation with a special operator, *Unless*, with peculiar properties. Some difficulties with *Unless* are described.

THE FRAME PROBLEM

This paper proposes a method for handling the *frame problem* in representing conceptual, or natural-language-type information. The method is part of a larger calculus for expressing conceptual information, called PCF-2, which is described in Sandewall (1972), and which is a modification and extension of Sandewall (1971a). The method proposed here is identical to the approach of mine which B. Raphael mentions at the end of his paper on the frame problem (Raphael 1971).

Previous work on the frame problem

The frame problem is the following: suppose we have a sequence of situations (in the sense of McCarthy 1968) in an ordinary-world-type environment, and that we know all 'free-will' actions that have been performed between these situations, that is, all actions which we do not wish to explain purely by laws of cause and effect within the environment. This information enables us to make deductions about properties of the environment in each situation. In practice, most properties stay set unless something changes them. For example, the color of an object usually stays the same when the object is

moved, and of course when it is left alone. However, it may occasionally change, and we might be able to realize this by a trivial deduction (for example, if the object was re-painted) or by a more complex one (for example, if the object was coated with fluorescent color and somebody blew a fuse so that the UV lamp went out). The problem then is to find a deduction mechanism which changes properties when they have to change, which retains them when they do not change, and which (this is perhaps the most difficult part) does this with a moderate number of axioms and moderate amounts of processing time.

This problem may seem odd at first, but it is a very real problem in the design of systems that reason about sequences of actions. This includes problem-solving systems (which are supposed to find a sequence of actions that leads to a given goal), but also some other cases. For example, an intelligent CAI system along the lines of Carbonell's (1970) system, but for history instead of geography, would certainly need to reason about sequences of actions, and therefore would need a solution to the frame problem. In this paper we are concerned with the frame problem for general use, as exemplified by the history application, but not with short-cut methods for problem-solving in simple environments.

Several previous authors have discussed the frame problem and proposed solutions for it, particularly Raphael (1971), Fikes and Nilsson (1971), and McCarthy and Hayes (1969, 1971a, 1971b). The work of Hewitt (1968–70) and the descendant work of Sussman and Winograd (1970) is also quite relevant to the frame problem, as has been pointed out, for example, by Hayes (1971b).

Let us very briefly comment on the proposed approaches from our viewpoint of 'general use'.

1. *The general frame axiom approaches* of McCarthy and Hayes (1969) and Hayes (1971a). In these approaches one attempts to write very general rules for when properties are retained, and their retention then has to be re-proved for every new situation. We doubt if it is possible to get a set of frame axioms with sufficient coverage, and we therefore reject these approaches.

2. *Consistency-based methods.* This approach is discussed by Hayes (1971b) and is based on work by Rescher (1964) and Simon and Rescher (1966). The basic idea is to remove properties only when they cause an inconsistency. Some conventions are needed for deciding which property is responsible for a detected inconsistency. We believe that this method could be useful, but that it will be very costly in computation time, so that a more immediate method is desirable.

3. *The* STRIPS *approach* of Raphael (1971) and Fikes and Nilsson (1971). Here facts are classified (on syntactic grounds) into 'primitive' and 'non-primitive'. Every action is characterized by a schema which specifies which primitive facts in the situation are added and deleted by the action. Other primitive facts are retained automatically. Non-primitive facts must in

principle be re-proved from the primitive facts in every new situation, although in practice one can design the program so that the deduction need not be re-performed.

One can think of cases for which this scheme is not sufficient, for example, Hayes' example (1971a): if a cup is standing on a saucer and the saucer is moved, the cup usually comes with it, but if only the cup is moved, the saucer (usually) does not come with it. We would like to add another example of a different nature. Let us make the reasonable assumption that friendship assumes that you are alive, that is, it is impossible for a dead person to have friends, or to have a dead person as a friend. We then want the 'friendship' property to be 'turned off' when a person dies. However, we do not wish to burden the schemas associated with actions like 'die', 'kill', and so on, with information regarding the cessation of friendship, and all other things that change at such a time. On the other hand, if 'to be a friend of' is to be a non-primitive, then we do not see what the supporting primitive property (-ies) could be. From such examples, we conclude that the STRIPS approach is probably limited to the problem-solving-type situations for which it is presently being used.

4. *The* PLANNER *approach.* Since PLANNER is (among many other things) a proposed programming language, any other approach would be a PLANNER approach in the sense that it could be implemented in PLANNER. However, there is one way of handling the frame problem, using the PLANNER primitives in a straightforward fashion, which we shall discuss here.

PLANNER enables the user to write rules which specify 'things that can be done'. Such a rule might, for example, correspond to a STRIPS schema. When the STRIPS schema adds a fact, PLANNER would add the corresponding fact to the data base using the primitive *thassert*. When the STRIPS schema states that a fact should be removed, the corresponding PLANNER rule would contain a command to remove the fact from the data base, using the primitive *therase*. Some care must be taken; for example, it is necessary to do the *therase*s of a situation transformation before the *thasserts*, so that fresh facts are not erased.

Although the PLANNER approach has several points in common with the STRIPS approach, the above two examples (cup and saucer, and dying friend) do not present any difficulties in themselves. Both of them can be handled by doing forward deduction of the form

$$\textit{therase}\ (A) \supset \textit{therase}\ (B).$$

Moreover, the programming language aspect of PLANNER makes it possible to counter any proposed, single counterexample, in the worst case by writing a piece of code. However, we depart from the PLANNER approach for another reason, which will be discussed in the next section.

EPISTEMOLOGY VS. HEURISTICS

McCarthy and Hayes (1969) discussed possible criteria for adequacy for a proposed notation, and made the following distinction: 'A representation is called epistemologically adequate for a person or machine if it can be used practically to express the facts that one actually has about the aspect of the world .... A representation is called heuristically adequate if the reasoning processes actually gone through in solving a problem are expressible in the language'. In our case, the 'aspect of the world' is of course the natural-language aspect, rather than, for example, the quantum physical aspect. Natural language itself is one epistemologically adequate representation, but (as discussed in Sandewall 1971b) we want another one which is more suitable for the task at hand, that is, question-answering, problem-solving, and other reasoning by computer.

Notice that the word 'heuristic' is here used in a slightly broader sense than usual. If we have a system which performs its reasoning by breadth-first search (which is not 'heuristic' in the ordinary sense), and if this and other search strategies are describable within the language, then the language is here termed heuristically adequate. However, the major reason for having a heuristically adequate system is of course to be able to communicate non-trivial heuristic guidance to the system.

The PLANNER approach to computer reasoning makes it a virtue to integrate epistemological and heuristic information, and the PLANNER language is the notation for expressing these together. We believe that this is permissible for problem environments of moderate complexity. However, for real-life problem environments, the task of writing a reasoning program is so complex that it is necessary to divide it into sub-tasks, and one very reasonable division is to study the epistemological problems first, and to add the heuristics afterwards, rather than try to do everything at once. This is particularly attractive since a set of rules *without* heuristic information (or other control information) is much less connected, and therefore much more modular, than the rules with heuristic information.

In this context, by epistemological information we mean a notation together with a set of rules (for example, logical axioms) which describe permissible deductions. The predicate-calculus notation proposed in our previous reports (1971a, 1971b, 1972) is intended to serve in this fashion. However, we stress that such systems are intended as *a basis for* programming in some programming language, and that it is essential that heuristic information is later added. Attempts to feed such systems into uniform-strategy theorem-provers (that is, to use them as *a substitute for* programming) are bound to fail for efficiency reasons.

The argument has sometimes been raised, at least in informal discussions at conferences, that epistemology *cannot* be separated from heuristics, that is, that in order to express facts in a convenient fashion, one must utilize statements about or make assumptions about when and how these facts are to be

used in the reasoning process. The discussion of approaches to the frame problem above might seem to verify that argument, since the PLANNER approach seemed the most satisfactory one. If we want to argue the distinction between epistemology and heuristics, we should therefore provide a mechanism which handles the frame problem *without* using heuristic information.

The present paper intends to do this – using a mechanism, the *Unless* operator, which is an extension to predicate-calculus notation, and which adequately handles the frame problem. At the same time, the *Unless* operator is independent of the theorem-prover or other interpreter that executes the search (in the data base), and of the heuristic information that governs this interpreter. We shall outline several different execution strategies, all of which are compatible with the intended meaning of the *Unless* operator. In that sense, our notation stays on the epistemological level, and is neutral with respect to heuristics.

THE HOST NOTATION

As was previously mentioned, our approach to the frame problem is part of a larger system for expressing conceptual information, called PCF-2. The full PCF-2 notation is too extensive to be described here, but for the present purpose it is sufficient to describe a subset.

We utilize the following sorts:

objects (which may be physical objects, or persons)
properties (e.g. 'red', 'angry', 'male')
situations (in McCarthy's sense of the word)
actions (e.g. 'to walk', 'to smoke', 'to smoke cigarettes in bed', 'to push', 'the robot pushing', 'to push a box', 'to push the box called :box12')

The following relations and functions are needed:

IS: object × property × situation
states that the object has the property in the situation.

INN: action × situation
states that the action occurs in the situation.

Case functions, of which there are several, one for each Fillmoresque 'case'. Their sorts are usually

$$\begin{cases} \text{action} \times \text{object} \rightarrow \text{action} \\ \text{action} \times \text{property} \rightarrow \text{action} \end{cases}$$

They are used to construct composite actions (for example, 'to smoke cigarettes' from simpler actions (for example, 'to smoke')

Succ: situation × action → situation
Maps a situation into that successor situation which results if the action is taken.

Additional relations and functions are needed, for example, to characterize the hierarchy of properties (human – mammal – animate . . .) or the succession ordering of situations, but they are not of interest here.

THE APPROACH TO THE FRAME PROBLEM

Given the system above, we have two versions of the frame problem, one for properties and one for actions. In the former, we want an object to retain a property (for example, 'red') until the property is explicitly changed (for example, by repainting). In the latter, we want an action to last (for example, 'John sleeping') until the action is explicitly discontinued by another action (for example, 'Peter waking up John'). Let us outline the solution to the property version of the problem.

We introduce a ternary relation

ENDS: object × property × situation

and a *frame inference rule* which with some simplification may be written as

$$\frac{\text{IS}(o, p, s),\quad \textit{Unless}(\text{ENDS}(o, p, \textit{Succ}(s, a)))}{\text{IS}(o, p, \textit{Succ}(s, a))}$$

The *Unless* operator makes this rule peculiar. The rule is intended to mean: 'if it can be proved that IS(o, p, s), and if it can not be proved that ENDS$(o, p, \textit{Succ}(s, a))$, then it has been proved that IS$(o, p, \textit{Succ}(s, a))$'.

The solution to the other, action version to the frame problem is analoguous.

This approach to the frame problem gains its strength (as compared to, for example, STRIPS) from the fact that one can make deductions to any depth using the predicate ENDS. Thus in the dying friend example, one could have a rule to the effect 'if x ends being alive, then x ends being a friend of y'. In the cup and saucer example, one would have 'if x supports y, and x moves to l, then y moves to l', plus 'if x moves, then x ends being where it was'. (Writing these rules out in the formalism is a trivial task.) Furthermore, this approach also makes it possible to do deductions backwards in time, such as 'since A is the case now, B must have been the case in the previous situation', where B or conclusions from B may then be used in *Unless* clauses. Such deduction may appear in history-type reasoning, although probably not in problem-solving applications.

THE *UNLESS* OPERATOR

The introduction of the *Unless* operator is a drastic modification of the logic. It even violates the extension property of almost all logical systems, which says that if you add more axioms, everything that used to be a theorem is still a theorem. However, it is not obvious whether this is serious. Our reason for using predicate calculus in the first place was that we wanted a *notation* in which to express conceptual information, and which could serve as a basis for a computer program that is expected to become large and complex. In other words, predicate calculus is only used for its syntax. The *Unless* operator extends the notation, but it is still reasonably clear how it is implemented in practice in a 'backward-search' algorithm. If we have the above frame inference rule of the form

$$\frac{A \quad \textit{Unless } B}{C}$$

and if we wish to prove C, we first make it a sub-problem to prove A. If we succeed, we then make it a sub-problem to prove B. If we succeed in this proof, then the proof of C did not succeed, and vice versa. Programming-wise, it should not be a problem that the proof of B might involve another *Unless*-clause. The fact that the search for a proof of B might have to be interrupted should not bother us too much. In an advanced system, the search for a proof for B might later be resumed in a final check-out of a proposed plan, or line of reasoning. This implementation of the *Unless* operator is similar to the PLANNER *thnot*, and in fact PLANNER would be a convenient (although expensive) implementation language for this system.

However, the fact that the *Unless* operator has some dirty logical properties should not be completely ignored. In one form or another, these difficulties are bound to appear in any implementation of an *Unless* operator. They are intrinsic and cannot be evaded, for example, by the choice of programming language. In this paper, we shall merely draw attention to some of these problems, without attempting to provide a solution. In the sequel, we shall permit *Unless* in wffs, for example, in axioms, with the obvious intended meaning.

First, consider the axioms

$$A$$
$$A \wedge \textit{Unless}(B) \supset C$$
$$A \wedge \textit{Unless}(C) \supset B$$

Clearly we cannot permit both B and C to be theorems simultaneously. Which of them is a theorem, if any? One possible approach to resolving the question is to use a precedence ordering on the axioms. Another approach is to restrict the structure of permitted axioms so that such situations can not occur. A third possibility is to bite the sour apple and accept that theoremhood is three-valued: theorem, not a theorem, or undetermined.

If the above backward-search algorithm is asked to prove B, it will create the sub-questions B and C alternatingly while digging down into the recursion. Its final answer (theorem or not theorem) will depend on which of the two sub-questions is at the break-off point. This is of course not satisfactory. One would like to have a definition of theoremhood which uses relatively conventional logical methods, that is, by specifying a procedure which will generate all theorems and only them. The practically useful backward-search algorithm would then be modified so as to approximate the theorem generator. However, it is not easy to set up such a procedure which also satisfies our intuition about how the *Unless* operator would behave. Consider, for example, the following seemingly very natural definition:

We define a restricted wff(rwff) as a two-tuple $\langle e, s \rangle$, where e is a wff and s is a set of wff. We extend all inference rules for wff,

$$e_1, e_2, \ldots, e_n \vdash e$$

into corresponding inference rules for rwff of the form:

$$\langle e_1, s_1\rangle, \langle e_2, s_2\rangle, \ldots, \langle e_n, s_n\rangle \vdash \langle e, s_1 \cup s_2 \cup \ldots \cup s_n\rangle$$

Moreover, we add the inference rule

$$\langle \mathit{Unless}(B) \supset e, s\rangle \vdash \langle e, s \cup \{B\}\rangle$$

(The notation is sloppy, but the intention should be clear.) We then specify the following inference mechanism for rwff:

1. From a set $A=\{a_i\}$ of axioms which are wffs, we construct the set
$$X_0=\{\langle a_i, \varnothing\rangle\}$$
where $\varnothing$ is the empty set.
2. For every X_i, we construct Y_{i+1} by adding the results of all possible one-step applications of inference rules.
3. From Y_{i+1}, we construct X_{i+1} by deleting all $\langle e, s\rangle$ such that $e' \in s$ where $\langle e', s'\rangle \in Y_{i+1}$.
4. For a given e: if there exists some s and K such that $\langle e, s\rangle \in X_k$ for all $k>K$, then e is a theorem.

With this procedure, neither B nor C in the above example would be theorems. Unfortunately, in the following example,

$$A$$
$$A \wedge \mathit{Unless}(B) \supset C$$
$$A \wedge \mathit{Unless}(C) \supset D$$
$$A \wedge \mathit{Unless}(D) \supset E$$

our intuition would permit E to be a theorem, but the above algorithm would not. It seems that more work is needed on this problem.

TIME-SEQUENTIAL DEDUCTION

In the face of the peculiarities of the *Unless* operator, it is tempting to use the selected frame inference rule in a deduction which proceeds 'forward' in time. Such a deduction might be performed as follows:

1. Collect all information about the initial situation. Make deductions from it, as long as the conclusions refer to the initial situation. (In practice, it may of course be necessary to interrupt this deduction.)
2. Determine all unconditional information (not relying on *Unless* conditions) about the next situation, for example, by using information about the action that led to this new situation.
3. Add the frame inference rule, and make forward deduction as far as possible about the new situation.
4. Transfer properties from the old situation, except in those cases when it is blocked by an 'ENDS' relation.
5. Remove all 'ENDS' relations. Then go to 2.

This procedure belongs to the same class of methods as STRIPS and the PLANNER method, whereas the general *Unless* deduction that was mentioned above is more similar to the consistency-based methods.

In a problem-solving situation, where we search a tree of possible futures for a good plan, we could strongly limit the forward deduction in step 3

during the planning phase. In the checkout phase, when we have found a plan, we would then go back and continue the deduction in step 3 in order to detect whether something could go wrong in the plan.

This time-sequential algorithm is not completely general, since it assumes that all deduction is performed within a situation, or from a situation to its successors. It will therefore not be sufficient if we require deductions 'backwards' in time, saying 'since *A* is the case in the present situation, *B* must have been the case in the previous situation'. Such deductions do not need to be common, but they do exist. We therefore believe that the time-sequential deduction is satisfactory in a planning phase, where one can ignore the backward-time deductions, but that a more general procedure, which can handle the full power of the *Unless* operator, is needed in the analysis of a given history, and possibly also in the checkout of plans.

Finally, let us remark that the *Unless* operator may be useful in some other cases besides the frame problem. For example, in handling hypothetical situations ('suppose you had been born two years later') we wish to assume all facts of our ordinary situation *unless* they are explicitly or implicitly changed by the hypothesis. There is some discussion about this in section 13 of Sandewall (1972). Here, again, a comparison with the consistency-based method of Rescher is relevant: the *Unless* operator is less elegant, but closer to a practical implementation.

Acknowledgements

This research was supported by the Swedish Natural Science Research Council under grant Dnr 2654-006.

REFERENCES

Carbonell, J.R. (1970) Mixed-initiative man-computer instructional dialogues. *BBN Report 1971*, Job no. 11399. Cambridge, Mass.: Bolt, Beranek & Newman Inc.

Fikes, R.E. & Nilsson, N.J. (1971) STRIPS: a new approach to the application of theorem proving to problem solving. *Art. Int.*, **2**, 189–208.

Hayes, P.J. (1971a) A logic of actions. *Machine Intelligence 6*, pp. 495–520 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Hayes, P.F. (1971b) The frame problem and related problems in artificial intelligence. *A.I. Memo 153*, Stanford Artificial Intelligence Project. California: Stanford University.

Hewitt, C. (1968) PLANNER: a language for manipulating models and proving theorems in a robot. *A.I. Memo 168*, Artificial Intelligence Project MAC. Cambridge, Mass: MIT.

McCarthy, J. (1968) Situations, actions, and causal laws. *Semantic Information Processing*, pp. 410–17 (ed. Minsky, M.). Cambridge, Mass.: MIT Press.

McCarthy, J. & Hayes, P.J. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pp. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Raphael, B. (1971) The frame problem in problem-solving systems. *Artificial Intelligence and Heuristic Programming*, pp. 159–69 (eds Findler, N.V. & Meltzer, B.). Edinburgh: Edinburgh University Press.

Rescher, N. (1964) *Hypothetical Reasoning*. Amsterdam: North Holland Press.

Sandewall, E. (1971a) Representing natural language information in predicate calculus. *Machine Intelligence 6*, pp. 255–77 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Sandewall, E. (1971b) Formal methods in the design of question-answering systems. *Art. Int.*, 2, 129–46.

Sandewall, E. (1972) PCF-2, a first-order calculus for expressing conceptual information. *Computer Science Report*. Uppsala: Computer Sciences Department, Uppsala University.

Simon, H.A. & Rescher, N. (1966) Cause and counterfactual. *Philosophy of Science*, 323–40.

Sussman, G.J., Winograd, T. & Charniak, E. (1970) Micro-Planner Reference Manual. *A.I. Memo 203*, Artificial Intelligence Project MAC. Cambridge, Mass: MIT.