How to Use What You Know

Carl Hewitt
M.I.T. Artificial Intelligence Laboratory

"A subject S understands knowledge K
if S uses K whenever appropriate."
Allen Newell

Abstract

This paper surveys the state of the art in the development of problem solving mechanisms for PLANNER-like formalisms. A major design goal for such formalisms is that *"for any given piece of knowledge there arc natural placet in the formatitm to procedurally embed the knowledge MO that it will He u$ed when it is appropriate and not used when it it inappropriate."* Although this goal is still very far from being achieved, considerable progress is being made. This paper contains no major conceptual innovations although it does describe and give the rationale for many of the design decisions that have been made In PLASMA [for PLANNER-like System Modeled on Actors) which is currently being implemented at MIT. as part of a Programming Apprentice project and for other applications. The paper tries to delineate major unsolved problems and make suggestions for how some of them may be attacked. Arguments are presented against the competing predicate logic paradigm for the implementation of knowledge-based systems.

Relationship to QA-4 and POPLER

Roth POPLER {for POP-2 PLANNER) (Davies 1972] and QA-4 (Rulifson et a! 1972) are basically cleaner *versions* of PLANNER 71 POPLER made important contributions in studying how to translate English quantifiers into a procedural form and in pointing out the ambiguity of the goal statement of PLANNER. A statement of the form (goat (on A B)) in PLANNER 69 could be satisfied either by demonstrating that A already is on B or by changing the world so that A is on B QA-4 drnves from scries of question answering programs done at SRI in the late 1960s and early I970Y The QA-2 and QA-3 [Green and Raphael 19713 systems were both resolution-based uniform proof procedure theorem provers. *QA-4* abandoned the resolution based theorem prover problem solving paradigm and adopted the procedural embedding of knowledge approach of PLANNER The most original contribution of QA-4 was Rulifson's development of a context mechanism.

QA-4 and POPLER also made some important improvements of the syntax of PLANNER-iike languages. PLASMA has continued in this tradition of making improvements to make PLANNER-like formalisms more readable by making them more "English-like". However, ft is important not to be misled into reading more into the symbols in the code than is really there. In every case the the semantics of the English symbols must be implemented elsewhere in other modules to realize the desired behavior. For example we shall glibly make assertions in the data base such as (A *is-on B)*. However, at some place [either explicitly or implicitly) code must be provided which specifies how actors created by the template (... is-on ...) behave in matching and how they index themselves in the data base. There is the potentiality for doing the indexing cleverly taking advantage of the fact that is-*on is* a transitive asymmetric relation. But in order to do so the specialized code must be made known to the (... it-on ...) template.

The major advance accomplished in PLASMA has been to explicitly base the semantics of a PLANNER-like formalism on actors. The actor semantics have resulted in a number of improvements in PLASMA over QA-4 POPLER, and CONNIVER

Programming Methodology

Actors make a contribution to the "declarative-procedure" controversy in that they subsume both the behavior of pure procedures (functions) and pure declaratives [data structures] as special cases. Discussions of the controversy that do not explicitly recognize the ability of actors to serve both functions may be doomed to sterility.

Actor semantics promotes a programming methodology which consists of the following activities:

Deciding on the natural kinds or objects to have in the domain.

Deciding for each kind of object what kind of messages it should receive.

Deciding for each kind or object what it should do when it receives each kind of message.

The above is a generalization of the programming methodology pioneered by SIMULA in which we have unified classes, procedures, and coroutines Once the above decisions have been made, it should be possible to directly implement the design directly without circumlocutions.

Unification of Pattern-Directed Invocation

We do not want to have to explicitly store every piece of knowledge which we have but would like to be able to derive *conclusions* from what *is* already known using procedures. The syntactic form *{when ... try ...*) as in *{when trigger* try body) or completely cquivalently the syntactic form *{to... try...*) as in (to trigger try body) creates a plan [high level goal-oriented procedure) that can be invoked by pattern directed invocation by a trigger which matches trigger There is a great deal of confusion in the literature about "demons". The phrase "completely cquivalently" above is meant to emphasize that there really is just one pattern directed invocation primitive. Implementing "antecedent" [working forward) plans entails facing the same issues as implementing *"consequent"* [working backward) plans. These problems involve doing retrieval, data base updates, and dealing with multiply applicable plans.

The following are kinds of plans which could be defined in terms of the above general pattern directed Invocation machinery. Additional kinds of plans can be defined as they are needed with the user defining their semantics by writing procedures for the triggers and bodies.

```
(to (demonstrate hypothesis) try body)
(when (asserting statement) try body)
(to (disachieve condition) try body)
(when (denying statement) try body)
(to (refute hypothesis) try body)
(to (find description) try body)
(to (achieve condition) try body)
(to (instantiate stereotype) try body)
(to (confront a-hypothesis with a-contradictory-hypothesis) try body)
```

The process of invoking plans in worlds is controlled by recommendations made at the site where a request Is made of the world to achieve a particular goal. We envisage that problem solving would begin in a world with an initial dass of plans, [n <u>many cases most of the plans that are used in the ultimate solution of the problem need to be constructed by other plans during the problem solving process.</u>

World Directed Invocation

World directed invocation [Hewitt et al. 1973, Stansfteld 1975, Freuder 1975) is invocation of procedures on the basis of a fragment of a micro-world instead of a single assertion or goal as in previous PLANNER-like fo'malisms. It is a generalization of pattern directed invocation [Hewitt 1969] such that more than one assertion is taken into account in directing the invocation. We foresee that the further development of world directed invocation and plans will play an important role in future development of PLANNER-like languages Within (he actor model of computation, the only way a piece of knowledge can be used is for actors, which have the knowledge, to be sent messages which cause them to act on it. We foresee that programming distribution networks in worlds to handle world-directed invocation will be important for effective modular distribution of knowledge. World directed invocation is still In a relatively undeveloped state by comparison with pattern directed Invocation which has been implemented in many PLANNER-like languages.

An important issue is how to maintain control over the invocation process. Two general mechanisms for this control are <u>recommendations</u> [Hawitl IJCAI-69, IJCAI-71) and <u>clustering</u> [Coldstein 1973, Stansfietd 1975, Marcus 1975] Clustering is the technique of gathering together plans that have overlapping domains of applicability. Suppose we have two plans plan$_1$ and plan$_2$:

```
[plan1 =                              ;plan1 is defined to be the following
    (to (achieve (=a is-on =b)) try
            body1))

[plan2 =                              ;plan2 is defined to be the following
    (to (achieve (=a is-on =b)) try
            body2))
```

Further suppose that if b has a clear top then plan$_2$ Is much better than plan$_1$. We could cluster the two plans into one as follows:

```
[plan3 =
    (to (achieve (=a is-on =b)) try
            (demonstrate (b has-a-clear-top)
                (then:
                    body2)
                (else:
                    body1))))
```

One of the earlier and better examples of clustering is found In Coldstein's design of geometry experts [each for a particular goal like congruence] which are responsible for choosing the best or at least the order of applying the plans for that goal.

An example of the use of a recommendation is the following:

```
(achieve (X is-on Y) (using: (plan1 else plan2)))
```

which attempts to achieve block X on block Y by first attempting to use plan$_1$ but if that doesn't work then using plan$_2$.

Nested Continuation Control Structure

The use of nested *"then:"* and *"else:"* continuations as in the above example together with complaint-departments seems to solve the scoping control problem which had been plaguing PLANNER-like languages for some time. Sandewall [1973] and a revised version of QA-4 independently developed nested continuation control structure for PLANNER-like languages. CONNIVER attempted to solve the scoping control problem by introducing possibility lists and Landin-style non-hierarchical gotos. However possibility lists proved to have several deficiencies They introduced side-effects into the basic communication mechanism between methods in CONNIVER which made it difficult for users to debug their programs since doing a try-noxt operation to print the next possibility destructively interferes with the operation of the programs being debugged.

"Their [Sussman and McDermott] *so*l*ution, to give the user accets to the implementation primitives of PLANNRR. i$ hotnever, something of a retrograde mep (what are CONNIVER*s semantic*?), although pragmatically* useful and important in *the shert term. A better solutian is to give the user access to a meaningful tes of primitive control abilities in* on *explicit rcpretentotional $ehem* concerned uith deductive control."*
Hayes: "Soma Problemt and Non-ProWamt in Ropratantation Thoory"

Nested continuation control structure gives us the ability to influence or control any decision to the extent we desire while still retaining the high level goal oriented nature of PLANNER. PLASMA is able to accomplish this by basing its semantics on actor message passing and slightly changing the syntax of PLANNER-71 The change in syntax provides us with natural places to incorporate the control Information and enables us to avoid the gratuitous side effects in PLANNER-71.

In terms of the actor model of computation, <u>control structure</u> is simply a pattern of passing messages CONNIVER represented a substantial advance over PLANNER-69 [implemented as MICRO-PLANNER by Sussman. Winograd, and Charniak] in increasing the generality of goal-oriented computations that can be performed. However this increase in generality comes at the price of lowering the level of the language of problem solving. It forces users to think In low level implementation terms such as "possibility lists" and "a-links". We propose a shift in the paradigm of problem solving to be one of a s<u>ociety</u> of <u>Individual</u>s <u>communicating</u> by <u>passing messages.</u> Within this semantic paradigm PLASMA acts as a flexible transparent medium for expressing desired behaviors without forcing the user to always think in low level implementation terms. It raises the level of problem solving while retaining the flexibility to control any decision by the modular addition of more knowledge

Complaint Departments

The "generate and test" method of Newell and Simon It a fundamental procedural schemata An Intuitively appealing extension to the generate and test method Is to allow the consumer to talk. back, to the generator with complaints, advice, and encouragement. PLANNER-69 had a mechanism for attempting to do this [called "failure messages"] that was so clumsy as to be essentially unuseable. In a very nice piece of work, Scott Fahlman improved on this mechanism to construct an extremely sophisticated blocks world construction program. His Idea was to construct "gripe handlers" which communicate by making use of assignments to the local variables of gripe handlers and Landin-style non-hierarchical gotos to transfer control. However, the gripe mechanism Is still awkward because of the necessity of side-effecting the handler which introduces the possibility of gripes accidentally clobbering each other or gripers accidentally clobbering the variables of other procedures. The message passing semantics of actor control structure seems to provide a fairly elegant simple mechanism for customers to talk back to generators. [The idea is an adaptation of the stream concept of Landin.] The customer receives a complaint department along with each new candidate produced by the generator. At a later point the customer can send a message to the complaint department that contains the complaint or advice plus any other actors that might be of use. These complaint departments are used in conjunction with the nested continuation control structure described above to help maintain the desired degree of control over the high level goal-oriented plans of PLASMA.

Frames

Three important **frame mechanisms** are:

plans: as in PLANNER-like languages, Newton's square root algorithm, Charniak's supermarket plan, Hewitt's birthday party plan, Schank's restaurant plan, etc.

circumstantial situations: as in a frame of reference in physics, a McCarthy frame, the assassination of Kennedy, a motion picture frame, etc.

stereotypical situations: as in the definition of the concept of a chair, Minsky's room frame, etc.

Of course the above fuuy classification does not Include many important aspects of frames. However, as a matter of research strategy, we would like to investigate how the above three framing mechanisms an co-exist in a single problem solving system. To this end we are implementing a mechanism which we call a stereotype to enable us to use these three framing mechanisms simultaneously. The stereotype mechanism attempts to deal with only a smalt number of the Issues that have been raised by frames [1975 TINLAP pre-prints) as a "*fremewerk /or reprinting knowledge*" [Minsk y 1974). By Implementing prototype systems using the stereotype mechanism, we hope to gradually discover how to bring the above frame mechanisms into active cooperation. We define a stereotype to consist of

acuuection of characteristic objects

characteristic relations for those objects

a set of **plans** invoked by world directed invocation
for transforming the relations

It seems that many of the behaviors attributed to frames by Minsky can be realized by stereotypes The characteristic objects of a stereotype correspond closely to the slots of a Minsky frame and the characteristic relations of a stereotype correspond to the constraints of a Minsky frame. Minsky calls simple unary characteristic relations markers Stereotypes communicate by making assertions in the data base and by world directed invocation.

We instantiate stereotypes somewhat differently from the way in which Minsky instantiates frames Our approach Is to plug in definite candidates for all the characteristic objects of a stereotype that we can and use anonymous objects [Herbrand, Skolcmj for the rest. Defaults are done as assertions tagged to indicate that they are defaulted so that they can be easily displaced if an anomaly develops Defaults as tags on assertions have an important advantage over the default values for the slots of a Minsky frame in that often a default relationship is known between two objects of a stereotype even though no default value is appropriate for either object. For example, there is a default relationship between the month of the date of the birthday party and the month of the birthday of the guest of honor. However, there is no particular month which is a very good default value for either. The use of anonymous objects in PLASMA enables us to reason explicitly about various possibilites using assertions and goals in the data base without complicating its identifier binding mechanism. Previous PLANNER-iike formalisms have placed the burden of doing the communication on the binding mechanism. This process of instantiation Is closely related to the process of fiirther specification emphasized by Newell and Moore in Merlin.

Situational Tags Instead of QA-4 Contexts

> ***"A situation s is the complete state of the universe at an instant of time... Since the universe is too large for complete description, we shall never completely describe a situation; we shall only give facts about situations. These facts will be used to deduce further facts about that situation, about future situations and about situations that persons can bring about from that situation.***
>
> ***We shall further assume that the laws of motion determine from a situation all future situations. [This assumption is difficult to square with quantum mechanics, and relativity tells us that any assignment of simultaneity to events in different places is arbitrary. However, we are proceeding on the basis that modern physics is irrelevant to common sense deciding what to do, and in particular is irrelevant to solving the 'free will problem'.]"***
>
> ***McCarthy and Hayes 1969***

Although we shall use tags on assertions and goals to relativize them to particular situations, hypotheses, and contexts, we do not want to make any of the global assumptions of McCarthy and Hayes All of our situations will be local. For example, whenever one situation precedes another, we shall allow for the possibility that there is a third distinct si'uation which Is neither before the former nor after the latter. For a discussion of the issues involved in this decision for the actor model of computation see Creif and Hewitt [1975}

The following fuzzy categories are examples of circumstantial situations: physical states, mental slates, logical hypotheticals, hypotheses, view points, goal situations, predictions, and defaults. The following are some simple examples of assertions that use such situational tags:

```
((A is-on B) in situation1)     (situation1 occurred-before fall-of-Rome
((A is-left-of B) in view5)
((John has $10^6) in dream7) ((Nixon is-innocent) in mind-of-Pat)
(¬(A is-on B) in goal5)        ((John is-at home) on January-1-1984)
```

PLASMA does not use the QA-4 context mechanism (Rultfson et al. 1972): instead it uses explicit tags in assertions to keep track of the state of affairs in various situations. One problem with the usual use of the QA-4 context mechanism is that the problem solver is forced to attempt to propagate all changes in the situation immediately on a frame shift since otherwise inconsistent information will be inherited from the previous situation. Another problem with the QA-4 context mechanism is that it is sometimes difficult to reason explicitly about various situations using it because situations (frames) are not explicitly part of the assertions and goals.

For example consider two contexts such that in the first Nell is standing cm the earth and in the second he is standing on the moon. This can be accomplished using contexts by generating two new contexts [call them context4 and conloxtg] and then asserting (Noil is-*standing-on* oarth) with-respect-to con1oxt$_4$ and asserting (Noil is-st*anding-on* moon) with-respect-to contoxt$_5$. However, consider the assertion that the weight of Neil in the first context is greater than his weight in the second This is not an assertion that can be made with respect to either context atone. Instead it seems that it is necessary to incorporate tags for both contexts into the assertion.

((wight Neil in context$_4$) > *{weight* Nail in context$_5$))

Events that are viewed from several different viewpoints can be difficult to handle. For example, consider the problem faced by Miss Marple in solving a murder mystery which happened at a party attended by several guests. Each guest has a fairly coherent story to tell but [at least] one of them is lying. Gerry and Marilyn are two guests [considered by Miss Marple to be generally reliable] who witnessed the shooting. Gerry's description is contained in contexl; whereas Marilyn's description is in contoxtg. We would like to construct a new context which inherits Information from both these descriptions. However, in the QA-4 context mechanism a sub-context inherits assertions only from the one stngle context of which it Is an extension created by the puth-eontoxt primitive Of course, there will be some slight discrepancies between Gerry's description and Marilyn's description even though both are "thoroughly reliable upstanding citizens" Methods for dealing with this problem are not well understood and will be discussed further later in the paper.

However, without the example of the QA-4 contexts to guide us, we could never have realized how to deal with these problems using situational tags. [The context mechanism In CONNIVER was modeled on the one in QA-4.]

It is easy to see how the abilities of QA-4 contexts can be implemented using situational tags As a first approximation In which erasing in contexts is not implemented, the following procedure will work.

```
[demonstrate-in-context ≡
    (to (demonstrate (=statement in =context)) try
        (find (context inherits-from =another-context)
            (then:
                (demonstrate (statement in another-context)))))]
```

Often in frame transitions, it is necessary to be able to make deductions in both directions across the transition. For example suppose that we are told the following Information:

Henry inherited one hundred thousond dotlon from hi$ parents Jill and Jack who died suddenly lenring no mill. The laws require that the ettate in such cases he divided evenly emong the children.

We can derive the following information for the Initial situations

```
((Henry is-alive) in s)
(((Henry inherits $100K) from Jill-and-Jack) in s)
(¬(Jill-and-Jack are-alive) in s)
((Jill-and-Jack are-parents-of Henry) in s)
((Jill-and-Jack left-no-will) in s)
```

Furthermore we need to know numerous rules such as the following:

```
(when (asserting ((Jill-and-Jack are-parents-of =person) in s)) try
        (assert ((person inherits $100K) from Jill-and-Jack) in s))

(∀ (≡> [=p =s1 =s2]
   (implies
        (∧
            ((p is-alive) in s2)
            (s2 is-after s1 is-after (birth p)))
        ((p is-alive) in s1))))
```

The next day, Henry met his older brother Paul (who is very miserly) in Times Square.

```
((Henry has-brother Paul) in s')
((Paul is-alive) in s')          (s' is (1 day) later-than s)
(Paul is-frugal)                 ((age Paul) > (age Henry))
```

From the above circumstances we can draw some reasonable hypothetical conclusions. Certain new information can be derived about s from s'.

```
(((Paul inherits $100K) from Jill-and-Jack) in s)
((Paul is-alive) in s]((Jill-and-Jack are-parents-of Paul) in s)
```

Now, additional information can be derived about s' from s.

```
((Paul has $100K) in s')
```

Notice how in this example information has migrated back and forth between s and s' in both directions.

The Blocks World

The blocks world has proved to be one of the most fruitful micro-worlds for research in artificial intelligence. It motivated the introduction of a large number of the important features of PLANNER-69 [MICRO-PLANNER) which was used to good advantage by Terry Winograd as an convenient high level goal oriented procedural formalism in which to translate natural language The following are simple properties of the blocks world expressed in the quantificational calculus. In order to be effectively and efficiently used for many purposes this knowledge needs to be procedurally embedded in various plans.

One of the major aims of the Programming Apprentice Project Is to scrutinize the relationships and dependencies engendered by these embedding!

```
[definition-of-has-a-clear-top ≡
   (∀ (≡> [=x =s]                                    ;for all x and s
         (iff
            ((x has-a-clear-top) in s)
                        ;x has a clear top in situation s if and only if
            (v                                                   ;either
               (x is the-table)                       ;x is the table or
               (¬ ((∃ (≡> [=y] ((y is-on x) in s))) in s)))))))]
           ;it is not the case that there is some y on x in situation s

[is-on-implies-is-above ≡
   (∀ (≡> [=x =y =s]                             ;for all x, y, and s
         (implies
            ((x is-on y) in s)                          ;if x is on y then
            ((x is-above y) in s)))))]                    ;x is above y

[transitivity-of-above ≡
   (∀ (≡> [=x =y =z =s]                       ;for all x, y, z, and s
         (implies                                                  ;if
            (∧
               ((x is-above y) in s)                   ;x is above y and
               ((y is-above z) in s))                     ;y is above z
            ((x is-above z) in s)))))]                 ;then x is above z

[above-is-not-reflexive ≡
   (∀ (≡> [=x =s]                                    ;for all x and s
         ((¬ (x is-above x)) in s)))]
                              ;it is not the case that x is above itself
```

Planning

We have found it useful To incorporate two types of worlds called Utopia [desired situation] and reality [achieved situation) in every problem solving situation. Our desire is to incorporate PLANNING into problem solving using "islands" [Mtnsky 1963) as stepping stones. The Utopia of one problem solving situation is taken as the reality of another. Sometimes working forward from what is known toward the goal is a good strategy. Sometimes working backward from the goal is a good strategy. Usually a combination of both is best.

An island is a problem solving situation for which it is not known whether it is achievable from reality or whether Utopia can be achieved from it; but there is reason to hope that both are in fact possible, it might be imagined that such islands are extremely difficult to find. But as Cerry Sussman has pointed out; such islands are proposed all (he time in PLANNER-like formalisms. In order to demonstrate that a block x is above another block z, it is sufficient to demonstrate that there is a block y such that x is above y and y is above z

```
[demonstrate-is-above ≡
                   ;define the plan demonstrate-is-above as follows
   (to (demonstrate (=x is-above =z) in =u from =r)
      (demonstrate ((x is-above =y) in =s from r)
         (then:
            (demonstrate ((y is-above z) in u from s)))))]
```

The plan demonstrate-is-above attempts to find an Island s with a block y such that it is known that x is above y and it can be demonstrated y is above z We will explain more sophisticated forms of island construction in examples given later in the paper.

The following are examples of convenient plans for use In our blocks world that procedurally embed some of the above knowledge:

```
[achieve-clear-top ≡      ;the plan achieve-clear-top is defined to be:
   (to (achieve: (=x has-a-clear-top) in =u from =r)
         ;to achieve that x has a clear top in utopia u from reality r
      try
      (either                                                  ;either
         (demonstrate ((x is the-table) in u)
                               ;demonstrate that x is the table in u
         (disachieve                                       ;or disachieve
            (∃ (≡> [=y]            ;that there is some block y such that
               (y is-on x)))))))]                              ;y is on x

[disachieve-on ≡                ;the plan disachieve-on is defined to be:
   (to (disachieve (=x is-on =y) in =u from =r) try
                  ;to achieve x is not on y in utopia u from reality r
         (achieve ((x has-a-clear-top) in =r' from r)
            (then:
               (achieve ((x is-on (not y)) in u from r'))))))]
                         ;achieve x being on some object which is not y

[achieve-on ≡
   (to (achieve (=x is-on =y) in =u from =r) try
      (achieve
         ((∧
            (x has-a-clear-top)
            (y has-a-clear-top)) in =r' from r)
         (then:
            (put x on y resulting-in u from r'))))]

[(put =x on =y resulting-in =u from =r) ≡
                 ;put x on y resulting in u from r is defined to be:
   (demonstrate
      (∧
         ((x has-a-clear-top) in r)
         ((y has-a-clear-top) in r))
      (then:
         (assert
            (∧
               (u results-from (put x on y in r))
               ((x is-on y) in u)
               ((x has-a-clear-top) in u)
               (¬ ((y has-a-clear-top) in u))))))]
```

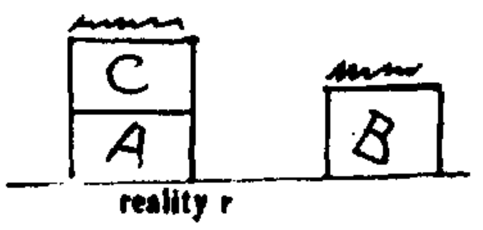
Constraint Analysis

The linear strategy for achieving a conjunct

$$(\wedge \text{ conjunct}_1 \ldots \text{conjunct}_k)$$

is to attempt achieve [and protect] each conjunct in turn in the order which they are presented. If this runs Into a protection violation, an attempt is made [on the basis of the kind of violation] to reorder the conjuncts and continue as before. The problem given below was invented by Allen Brown as an example of a simple blocks world problem that cannot be solved using the "linear" strategy without backtracking trying first one conjunct first then the other. We present this as a very simple illustration of how planning using "islands** can help keep the problem solving constrained and avoid search. None of this Is meant as a criticism of Sussman who designed and partly Implemented a really interesting problem solving system. His work inspired us to further develop the constraint analysis approach which Is a variant of the "means-ends" analysis used by Newell and Simon. The initial situation [reality] r is

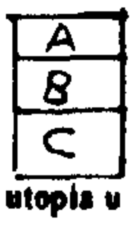
```
((C has-a-clear-top) in r)
((B has-a-clear-top) in r)
((C is-on A) in r)
((A is-on the-table) in r)
((B is-on the-table) in r)
```



reality r

The desired utopia u is

```
((A is-on B) in u)
((B is-on C) in u)
```



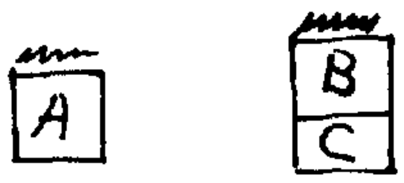
utopia u

<u>It is very desireable for assertions about utopia [the goal situations) as well as reality [the achieved situations] to be explicitly placed in the data base so as to be readily available to plans working forwards and backwards.</u> Among other things, this implements a version of "continuously available output from processes" which has been advocated by Bobrow and Norman.

Solution Working Backwards from Utopia

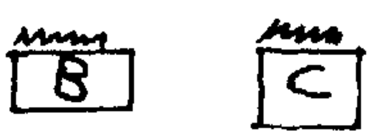
Constraint analysis can be used to straightforwardly solve the Brown blocks world problem as follows. Clearly, the Initial state is not a solution to the problem, tie. u Is not *r)* because ((C is-above A) in r) is incompatible with ((A is-*above* C) *in* u) The easiest way to solve the problem is by working backwards from Utopia u doing constraint analysis. <u>Constraint Analysis</u> is a planning technique in which abstract reasoning is used to derive the necessary structure of the possible solutions to a problem. In <u>working backwards from a situation, attempt to eliminate the</u> c<u>onstraints that don't hold in reality.</u> If Utopia u is to be achieved from a previous state u_1, it must be that the following holds about u_1

```
(u results-from (put A on B in u1))
((A has-a-clear-top) in u)
((B is-on C) in u1)
((¬ (B is-above A)) in u1)
((¬ (C is-above A)) in u1)
((A has-a-clear-top) in u1)
((B has-a-clear-top) in u1)
```



Again it is easy to see that the problem is not solved [u_1 is not r]. If utopia u_1 is to be achieved from a previous state u_2, it must be that the following holds about u_2:

```
(u1 results-from (put B on C in u2))
((¬ (C is-above A)) in u2)
((¬ (C is-above B)) in u2)
((B has-a-clear-top) in u2)
((C has-a-clear-top) in u2)
```



The problem is still not solved for the same reason. [Note that u_2 must have been obtained from some previous situation u_3 by moving C since this is the only way to eliminate the constraint that It is not the case that C Is above A]

```
(u2 results-from (put C on new-support-for-C in u3))
((C is-on new-support-for-C) in u2)
((C has-a-clear-top) in u3)
```

where <u>now-support-tor-C</u> is some position. Notice how limited and abstract the knowledge of Utopia u_3 has become in the course of abstractly planning backwards. Now it seems possible that w_3 might be r. So create a new hypothesis h.

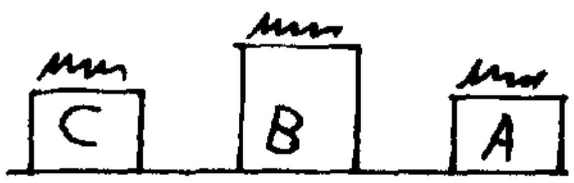
```
((u3 = r) in h)
((new-support-for-C ≠ A) in h)        ;since ¬((C is-above A) in u2)
((new-support-for-C ≠ B) in h)        ;since ¬((C is-above B) in u2)
((new-support-for-C ≠ C) in h)
                              ;since you can't put something on itself
```

Having nothing else to choose for <u>new-support-for-C</u>, choose the table:

```
((new-support-for-C = the-table) in h)
```

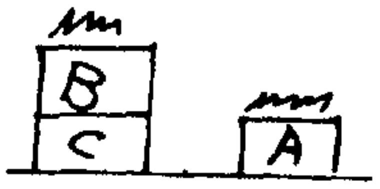
Now reason forward from r using the <u>plan</u> constructed by reasoning backwards from u. The first step is to construct a world r_1 from r:

```
(r1 results-from (put C on the-table in r))
((u2 = r1) in h)
((C is-on the-table) in r1)
((B is-on the-table) in r1)
((A is-on the-table) in r1)
((A has-a-clear-top) in r1)
((B has-a-clear-top) in r1)
((C has-a-clear-top) in r1)
```

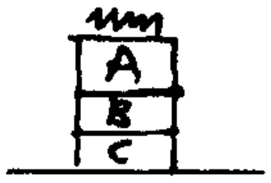


Everything seems to be going well, so attempt the next step in the tentative plan which involves generating a world r_2 from r_1:

```
(r2 results-from (put B on C in r1))
((u1 = r2) in h)
((C is-on the-table) in r2)
((A is-on the-table) in r2)
((B is-on C) in r2)
```



```
(r3 results-from (put A on B in r2))
((u = r3) in h)
((B is-on C) in r3)
((A is-on B) in r3)
```



We have achieved our goal in r_3:

```
((∧ (A is-on B) (B is-on C)) in r3)
```

The hypothesis h has led to the following plan for solving the problem:

```
(place (C on the-table)
  (then:
    (place (B on C)
      (then:
        (place (A on B))))))
```

The above solution can be varlabalized using procedural abstraction [I lewitt 1971; Sussman 1973] into a procedure as follows

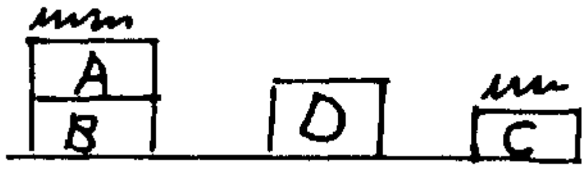
```
[to-achieve-three-stack_1 =
    (to-achieve
        ((∧ (=x is-on =y) (=y is-on =z))
            from (=r containing
                        ((z has-a-clear-top)
                         (y has-a-clear-top)
                         (z is-on x))))
        try
        (place (z on the-table)
            (then:
                (place (y on z)
                    (then:
                        (place (x on y))))))))]
```

Notice that in the above example that the plan to-achieve-three-stack$_1$ which we generated does not have as pre-requisites the following irrelevant incidental features of the original problem specification: (x is-on the-table) and (y is-on the-table). The reason is that the above features are never relied upon in the symbolic evaluation of the plan solution.

Working Backwards then Forwards

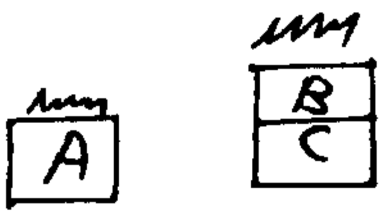
We would like to consider the problem of realizing the same utopia u from a different initial situation s described below:

```
((D is-on the-table) in s)
((A is-on B) in s)
((B is-on the-table) in s)
((C is-on the-table) in s)
((A has-a-clear-top) in s)
((C has-a-clear-top) in s)
```



The analysis proceeds much as before in terms of generating utopia u_1

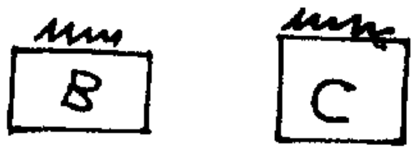
```
(u results-from (put A on B in u_1))
((A has-a-clear-top) in u_1)
((B is-on C) in u_1)
((¬ (B is-above A)) in u_1)
((¬ (C is-above A)) in u_1)
((A has-a-clear-top) in u_1)
((B has-a-clear-top) in u_1)
```



utopia u_1

and then utopia u_2.

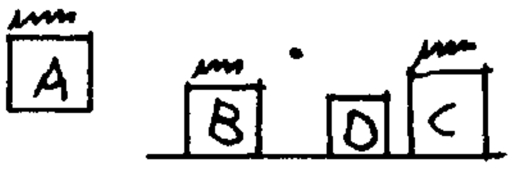
```
(u_1 results-from (put B on C in u_2))
((¬ (C is-above A)) in u_2)
((¬ (C is-above B)) in u_2)
((B has-a-clear-top) in u_2)
((C has-a-clear-top) in u_2)
```



utopia u_2

Comparing the new utopia with the original reality s, it is not obvious how to further constrain the structure of the solution. However, notice that u_2 is compatible with t except that B must have a clear top in u_2 whereas A is on B in s Thus there is a potentiality that u_2 can be reached from s by clearing the top of B in s to produce a new situation s_1. In working forwards from a situation, attempt to satisfy the constraints that hold in Utopia. In order to do this A must be put some place which we shall denote by new-tupporl-for-A

```
(s_1 results-from (put A on new-support-for-A))
((A is-on new-support-for-A) in s_1)
((B is-on the-table) in s_1)
((C is-on the-table) in s_1)
((A has-a-clear-top) in s_1)
((B has-a-clear-top) in s_1)
```



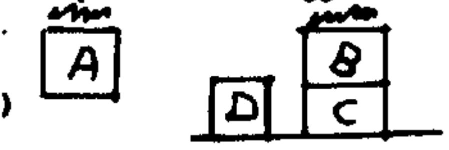
There are no readily discernible incompatibilities between u_2 and s_1, so create a hypothesis k as follows:

```
((u_2 = s_1) in k)
```

We immediately conclude:

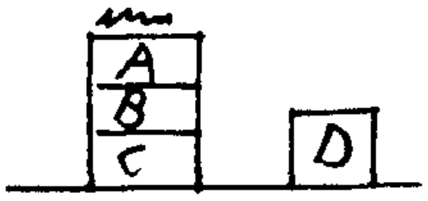
```
((C has-a-clear-top) in s_1)          ;since ((C has-a-clear-top) in u_2)
((new-support-for-A ≠ C) in k)
((new-support-for-A ≠ A) in k)
((new-support-for-A ≠ B) in k)
```

Our hypothesis k gives us a tentative plan to be debugged. We first generate situation s_2 from s_1.



```
(s_2 results-from (put B on C in s_2))
((u_1 = s_2) in k)
```

```
(s_3 results-from (put A on B in s_2))
((u = s_3) in k)
((A is-on B) in s_3)
((B is-on C) in s_3)
```



Thus the hypothesis k has led to a plan for solving the problem:

```
(place (A on (and (not A) (not B) (not C)))
    (then:
        (place (B on C)
            (then:
                (place (A on B))))))
```

Again by procedural abstraction we have:

```
[to-achieve-three-stack_2 =
    (to-achieve
        ((∧ (=x is-on =y) (=y is-on =z))
            from (=r containing
                        ((z has-a-clear-top)
                         (x has-a-clear-top)
                         (x is-on y))))
        try
        (place (x on (and (not x) (not y) (not z)))
            (then:
                (place (y on z)
                    (then:
                        (place (x on y))))))))]
```

A really interesting area for future research is how to recognize that both the plans to-achieve-three-stack$_1$ and to-achieve-three-stack$_2$ are instances of the more general plan "*Build a stack from the bottom up*". An important research area for the future is that of integrating plans using clustering, variabalization, and other yet to be discovered techniques.

Predicate Logic as Programming Language

In this paper we oppose the thesis that "*predicate logic is a useful and practical, high-level, non-deterministic programming language with sound theoretical foundations*" [Robert Kowalski in

"Predicate Logic as Programming languagal The deficiencies of the predicate calculus as a programming language are traced back to the tmtli-tlieoretic [Tarski 1930] semantics of logic. The nub of (he difficulty seems (o be the way In which meaning is defined, *in* that axioms may convey truth but not pragmatic information We explain how the behavioral semantics alleviate some of the difficulties with truth-theoretic semantics. This work is presented using Rotors, a semantic concept in which no active process is ever allowed to treat anything as an object; instead a polite request must be extended to accomplish what Is desired.

Necessity of a. Unified Formalism

For some years now we have been working to achieve the goal of a unified formalism and semantics for knowledge based systems. The record of our progress is published in the Proceedings of the International Joint Conferences on Artificial Intelligence beginning with the first conference in 1969. In the course of this research we have developed the Thesis of Procedural Embedding of Knowledge The useful knowledge off a domain is intrinsically bound up with the specialized procedures for its use. An important corollary is that the fundamental technique for the representation of knowledge is Procedural Knowledge Base Construction

> *"More specifically: I uill argue that computation is best regarded as a proem* of controlled deduction. It will he further argued that the two otpeett specifying the control mechaninm) of programming are heat tcftftraird explicirtly, as the kindt of language involved have quite dittinct $emanticM."*
>
> Pal Hayes: "Computation and Deduction"

A satisfactory theory for the representation of knowledge should have one unified totally integrated formalism and semantics. For example we should not have one formalism and semantics for expressing declaratives and a separate [as proposed by Hayes] formalism and semantics for expressing procedures. Having two formalisms [with different semantics] erects an arbitrary pernicious barrier between two ends of a useful continuum. One technique which shows great promise toward automating the development of procedural knowledge base acquisition is that of Progressive Refinement of Plans [Dljkstra, Wirth 1971; Cheatham and Wegbreit 1973; Hewitt 1971, 1975} This technique involves the transition from very high-level goal-oriented procedures to more particular efficient implementations for special circumstances. As such it spans the spectrum from relatively domain-independent goal-oriented (declarative?) modes of computation to particular efficient [procedural] modes The facility with which Progressive Refinement of Plans can be automated depends on having one formalism and semantics to span the above spectrum

Problems with Truth-Theoretic Semantics

> *"The Tarskian semantic theory of PC* (predicate calculus) *and its relatives are the only precise account we have of how a language can refer to extralinguistic entities."*
>
> **Pat Hayes: "Computation and Deduction"**

The definition of TRUTH as formalized by Tarskl for the quantificational calculus is one of the crowning achievements of mathematical logic It has clarified the semantics of ordinary mathematical theorems and led to the development of truth-theoretic model theory which is a flourishing mathematical field in its own right We contend that it Is less satisfactory as a semantic base for a theory of action and change Tarskian semantics characterizes predicate logic in the following sense: Goclel proved that for the first order quantificational calculus that the class of sentences deducible by the classical laws of logic is exactly the same as the ones which are true by Tarskl's definition The notion of truth formalized by Tarski is very smooth but we contend that it glosses over semantic distinctions that are crucial to the development of knowledge-based systems

The deductions of plans in a PLANNER-like formalism have the potential to *carry* more conviction [in the sense of Richard Weyhrauch and other logicians] than proofs in the classical predicate logic [For example a proof of a formula of the form (□ *imptice ψ)* carries conviction if it demonstrates a "causal" connection from □ to ψ.) This is because our minds are better at grasping constructive relationships between plans than global relationships established by asserting that a class of axioms is true or consistent Two plans can affect each other only if there is a behavioral causal connection of "wheels and. cogs". We seem to be able to design, control and debug plans better than axioms

In the rcal world no formal principle is universally literally true Any rule can be stretched beyond its range of applicability to produce contradictions Except in mathematics all useful principles are hemmed in with Qualifications, hedges, and caveats Knowledge is better expressed procedurally in the form of tough minded skeptical plans that actively attempt to guard against their misuse The business of actively searching out hidden anomalous situations is as important as the business of making deductions that carry conviction Indeed, much of the conviction carried by a set of plans is dependent on their ability to withstand repeated, meticulous, rigorous attempts at refutation from as many directions of attack as can be conceived No principles [including the classical laws of logic) are sacred against criticism when anomalies are detected.

No satisfactory method for testing the consistency of non-trivial sets of axioms in the quantificational calculus has been developed For example inconsistent formulations of the "Blind Hand Problem" [a particular toy robot problem] have been produced in the quantificational calculus Their inconsistency has been discoveted almost by accident as proofs by contradiction get shorter and shorter until the negation of the desired consequence is found to be superfluous to the proof! People are quite tolerant of minor inconsistencies The inability of the quantificational calculus to tolerate any inconsistency at all in formulating problems is a sign of excessive semantic rigidity An example of the kind of problem engendered by this rigidity when dealing with common sense knowledge is given below. In general we feel that a contradiction is evidence for a bug in one's plans or world model and that to satisfactorily resolve the bug, It may be necessary to examine al[the assumptions being made instead of oniy tne most recent one. Currently mere are no guuu ways to debug classes of axioms, whereas there is a well established and rapidly developing technology for debugging procedures [Miniky and Papert 1972, Sussman 1974; Goldstein 1974; Hewitt and Smith 1975].

Behavioral Semantics for Logic

> *"To defend first-order logic is unfashionable: however, I do want to emphasize that it is the semantics of predicate logic which I wish to preserve."*
>
> **Hayes: "Some Problems and Non-Problems in Representation Theory"**

We agree with the above comment of Hayes (except that we have a different notion of semantics for the language of the predicate calculus) The way in which the rulrs of natural deduction follow from the procedural mechanisms of PLASMA, demonstrates how deduction can usefully be considered to be a special case of computation. Furthermore, by implementing the useful part of the behavior of the quantificational calculus in PLASMA, we will obtain a behavioral semantics of the quantificational calculus from the underlying actor semantics of PLASMA Consider a formula of the form $\forall x.\Phi[x]$ which means that for every x we have that $\Phi[x]$ is the case The procedural meaning of the formula consists of plans for how it can be used. The plans we are attempting to develop for V and the other logical operators are reminiscent of natural deduction except that we have four introduction and elimination rules [demorwtrate, refute, assort, and deny) to give us more flexibility in dealing with negation. An example is the following plan:

```
[to-demonstrate-a-∀ ≡   ;the plan to-demonstrate-a-∀ is defined to be
    (to (demonstrate (∀ =Φ) in =u from =r) try
                   ;to demonstrate (∀ Φ) in in utopia u from reality r try
       (let                                                          ;let
          {[g = (anonymous-object-in r)]}
                                          ;g be an anonymous object in r
          (demonstrate ((Φ g) in u from r))))]
                              ;attempt to demonstrate Φ of g in u from r
```

In addition we include the nonlogical operators achieve and disachieve to give us similar flexibility in dealing with state changes. The following plan is an example:

```
[to-disachieve-a-∀ ≡
    (to (disachieve (∀ =Φ) in =u from =r) try
       ;to transform reality r so that (∀ Φ) does not hold in utopia u
       (disachieve ((Φ (ϵ (objects u))) in u from r))))]
                   ;attempt to disachieve Φ of some object in u from r
```

Our intent in attempting to formulate plans like the above Is to try to characterize the useful behavior of ∀, ∃, ∨, ∧, implies, and ¬. We do not intend for them to be used as the code to define a general uniform proof procedure. Instead they are intended to be used as template schemata in progressive refinement to construct more specialized domain dependent plans. The classical rules of logic are intended to generate all the true sentences from a given class of true sentences. Our behavioral definitions are intended to define the justifiable behavior of the logical operators that carries conviction.

Garbage in--Garbage out

In the quantificational calculus from a sentence of the form $(\wedge\ \underline{\Psi}\ (\neg\ \underline{\Psi}))$, every statement is deducible no matter how nonsensical! The following plan schemata realizes this behavior [although in our view this behavior is usually quite harmful]:

```
[gigo ≡
    (to (demonstrate (=Φ in =u from =r)) try
       (demonstrate
          ((∧ =Ψ (¬ =Ψ)) in u from r)))]
```

Inconsistencies in Common Sense Knowledge

> *"All questions concerning logical implication in first order logic can be replaced by questions concerning unsatisfiability of sentences in clausal form."*
>
> Robert Kowalski: "Predicate Logic as Programming Language"

The gigo plan graphically illustrates a fundamental problem with the truth-theoretic semantics for the predicate calculus The presence of the gigo rule in the quantificational calculus casts a shadow of doubt on the reliability of all of the computations of the system. For example the following set of assertions [McDermott 1973] is not consistent However. a problem solving system should not be able to conclude that 1+1 ■ 3 from the inconsistencyl

```
(birds fly)                       (penguins ⊂ birds)
(robins ⊂ birds)                  (penguins never-are robins)
(ostriches ⊂ birds)               (penguins don't-fly)
(Fred lives-in the-zoo)           (penguins never-are ostriches)
(robins fly)                      (the-zoo is-located-in Sydney)
(Fred doesn't-fly)                (Fred ϵ birds)
(ostriches don't-fly)
```

A more reasonable conclusion is that Fred might be an ostrich or a penguin. We feel that this problem of inconsistency is ripe for further research along the lines of McDermou [19731 Sussman [1974], Goldstein [19743, Kuipers [1974] Rubin [19751 Smith and Hewitt [19751 Erman and Lesser [19751 and McLennan [1975].

Every dubitable assertion in the data base needs to have explicitly recorded exactly which rules and assertions were used in its derivations. When an anomaly is detected particular specialized diagnosing routines must be invoked to deal with the confrontation.

Indireot Proof

Indirect proof is often a very useful technique in problem solving. The classical natural deduction rule for indirect proof takes the following form:

```
[indirect-proof ≡
    (to (demonstrate ((¬ =Φ) in =u from =r)) try
       (let
          {[r' = (extension-for-contradiction r)]}
          (assert (Φ in r')
             (then:
                (demonstrate
                   ((∧ =Ψ (¬ =Ψ)) in r'))))))]
```

However, it is easy to generate gigo using classical-negation and indirect proof. We are working to develop a more limited form of indirect proof in which the contradiction mutt depend in an essential way on the hypothesis being refuted. The behavioral semantics of *depending-on* needs to be carefully worked out to produce a deduction system that more nearly captures the logicians' notion of conviction as discussed above. Again a Key technique for attacking the problem is to explicitly record for every assertion the rules and hypotheses used in its derivations

Further Work

The PLASMA system described in this paper is currently being implemented at the MIT Artificial Intelligence Laboratory. The best version which currently runs was coded in LISP by Howie Shrobe A better humanly engineered version has been designed and coded by Carl Hewitt in PLASMA with the extensive aid of Marilyn McLennan. The new implementation has been translated into LISP by the members of a seminar this spring with the following participants: Russ Atkinson, Mike Freiling, Kenneth Kahn, Marilyn McLennan, Keith Nishihara, Howie Shrobe, Kathy Van Sant, and Aki Yonezawa The translation was made possible by LISP macros written by Russ Atkinson which makes the LISP code of the implementation closely resemble the PLASMA which it Implements A rough draft of a primer by Brian Smith and Carl Hewitt for the new Implementation exists

Next fall the implementation will be used as a basis on winch to build the higher level problem solving mechanisms described in this paper in order to implement some knowledge-based problem solving systems One project is the implementation of a Programming Apprentice The emphasis thus far in the project has been the elucidation and formalization of the semantic principles involved Procedures for analysis have been chosen for their ability to illustrate particularly difficult semantic problems. Examples that have been analyzed are pure queues [Hewitt and Smith], hash tables [Rich and Shrobe] impure queues (Yonezawal and various versions of the readers-writers problem [Greif] Each of these- examples is about one page of code in a high level language. But it takes an average programmer a half hour or so to really understand each one if it is being encountered for the first time We would like to continue this approach by analyzing even more difficult examples such as the disk directory system of a timesharing system and the Steele-Dijkstra parallel garbage collection algorithm. However, in addition we plan to take a larger system consisting of about ten or fifteen pages of code in order to test our techniques on whole systems. In this regard, following a suggestion of Michael Dertouzos. we plan to try the programming apprentice on a budgetary assistant program that aids in the construction and balancing of budgets.

Acknowledgements

This paper is a greatly elaborated written version of a talk delivered at the "Special Session on Formalisms for Artificial Intelligence" at IJCAI-73 at Stanford in August 1973 The research reported in this paper was sponsored by the MIT Artificial Intelligence and Laboratory and Project MAC under the sponsorship of the Office of Naval Research

Writing this paper would not have been possible without extensive conversations with Richard Weyhrauch. Candy Rullwinkle, Kenneth Kahn, Marilyn McLennan, Keith Nishihara. and Guy Steele made helpful comments and criticisms which considerably improved the intelligibility of this paper. Mitch Marcus suggested an improved title for the paptr and also made very helpful suggestions. Conversations with Allen Brown, Martin van Emden, Pat Hayes, Bob Kowalski, Marvin Mtnsky, Allen Newell. Earl Sacerdoti, Gerry Sussman, and Richard Waldinger have proved to be of great value in writing this paper. Although this paper has emphasized the deficiencies of truth-theoretic semantics for knowledge-based programming, we have learned a great deal from studying systems for attempting to use predicate logic as a problem solving formalism and interacting with the implcmentors of such systems Ira Goldstein, Howie Shrobe, and Gerry Sussman contributed valuable suggestions and criticisms on how to clarify the goals of this paper.

Bibliography

Davie., D J M. "POPLER: A POP-2 PLANNEIT MIP-89. School of A-I. University of Edinburgh.

Davie*, D . J . M. "Representing Negation in • PLANNER System" Proceedings el AISB Cenference July, 1974. Svssex.

Erman, L. and Lesser, V. "A Multi-Level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge. CMU. March, 1975.

Fahlman, S. "A Planning System for Robot Construction Tasks" M.I.T. A.I. TR-283. May, 1973.

Freuder, E. "Suggestion and Advice" M.I.T. A.I. Vision Flash 43. March, 1973.

Greif, I. "Semantics of Communicating Parallel Processes" Ph.d.. M.I.T. September, 1975.

Greif, I. and Hewitt, C. "Actor Semantics of PLANNER-73" Proceedings of ACM SIGPLAN-SIGACT Conference. January, 1975.

Goldstein, I. "Elementary Geometry Theorem Proving" M.I.T. A.I. Memo 280. April, 1973.

Goldstein, I. P. "Understanding Simple Picture Programs" Proceedings of AISB. Sussex. July, 1974.

Hayes, P. J. "Computation and Deduction" Proc. MFCS Symposium. Czech. Academy of Sciences. 1973.

Hayes, P. J. "Some Problems and Non-Problems in Representation Theory" Proceedings of AISB. Sussex. July, 1974.

Hewitt, C. "PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot" IJCAI-69. Washington, D. C.

Hewitt, C. "Procedural Embedding of Knowledge in PLANNER" IJCAI-71. London. Sept, 1971.

Hewitt, C. "Procedural Semantics" in Natural Language Processing Courant Computer Science Symposium 8. Algorithmics Press. 1971.

Hewitt, C. et. al. "A Universal Modular ACTOR Formalism" IJCAI-73. Stanford. Aug, 1973.

Hewitt, C. and Smith, B. "Towards a Programming Apprentice" IEEE Journal of Software Engineering. March, 1975.

Hewitt, C. "STEREOTYPES as an ACTOR Approach Toward Solving the Problem of Procedural Attachment in FRAME Theories" TINLAP Pre-prints. June 10-13, 1975. M.I.T.

Kowalski, R. "Predicate Calculus as Programming Language" IFIP-74. Vol. 3. pp. 569-574.

Kuipers, Ben. "An Hypothesis-Driven Recognition System for the Blocks World" MIT-AI Working Paper 83. March, 1974.

Landin, P. J. "A Correspondence Between ALGOL 60 and Church's Lambda-Notation" CACM, February, 1965.

Marcus, Mitch. "Diagnosis as a Notion of Grammar" TINLAP Pre-prints June 10-13, 1975. M.I.T.

McCarthy, J. and P. Hayes, "Some Philosophical Problems From the Standpoint of Artificial Intelligence" Machine Intelligence Vol. 4. American Elsevier Publishing Company. New York. 1969.

McDermott, D. "Assimilation of New Information by a Natural Language Understanding System" MIT AI Memo 298. 1974.

McLennan, M. "Hypotheses and Refutations as an Aid to Computer Vision" Thesis Progress Report. Edinburgh. June, 1975.

Minsky, M "Steps Toward Artificial Intelligence" Proceedings of Institute of Radio Engineers. Jan. 1961.

Minsky, M. "A Framework for Representing Knowledge" MIT AI Memo 306. June, 1974.

Moore, J. and Newell, A. "How can MERLIN understand?" Department of Computer Science. CMU. 1973.

Newell, A. and Simon, H. Human Problem Solving Prentice-Hall. 1972.

Rich, C. and Shrobe, H. "Understanding LISP Programs: Towards a Programmers Apprentice" M.I.T. A.I. Working Paper 82.

Rubin, A. "Hypothesis Formation and Evaluation in Medical Diagnosis" M.I.T. A.I. TR-316. Jan. 1975.

Rulifson Johns F., Derksen J. A., and Waldinger R. J. "QA4: A Procedural Calculus for Intuitive Reasoning" Ph. D. Stanford.

Sacerdoti, E. "Planning in a Hierarchy of Abstraction Spaces" IJCAI-73.

Sandewall, E. "Conversion of Predicate-Calculus Axioms, Viewed as Non-Deterministic Programs, to Corresponding Deterministic Programs" IJCAI-73.

Stansfield, J. L. "Programming a Dialogue Teaching Situation" Ph.D. Thesis. University of Edinburgh. January, 1975.

Sussman, G. J. "The Virtuous Nature of Bugs" Proceedings of AISB. Sussex. July, 1974.

Tate, A. "INTERPLAN: A Plan Generation System Which Can Deal With Interactions Between Goals" MIP-R-109. Edinburgh. December, 1974.

Warren, D. H. D. "WARPLAN: A System for Generating Plans" Dept. of Computation Logic Memo No. 76. June 1974.

Winograd, T. "Five Lectures on Artificial Intelligence" Stanford A.I. Memo. September, 1974.

Wirth, N. "Program Development by Stepwise Refinement" CACM. Vol 14. pp. 221-227. 1971.