ANNOTATED PRODUCTION SYSTEMS
A MODEL FOR SKILL ACQUISITION

Ira P. Goldstein and Eric Crimson
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
February 1977

Abstract

Annotated Production Systems provide a procedural model for skill acquisition by augmenting a production model of the skill with formal commentary describing plans, hugs, and interrelationships among various productions. This commentary supports efficient interpretation, self-debugging and self-improvement. The theory of annotated productions is developed by analyzing the skill of attitude instrument flying. An annotated production interpreter has been written that executes skill models which control a flight simulator. Preliminary evidence indicates that annotated productions effectively model certain bugs and certain learning behaviors characteristic of student.

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643, and in part by the Naval Training Equipment Center under contract N61339-76-C-0046.

1. Introduction

Productions systems have many strengths as a model of human problem solving — modularity, simple control, generality; but they are not sufficient. We argue for this position by analyzing the skill of flying an airplane under instruments. A production model will bo defined and its limitations considered. These limitations will involve inefficiencies in dealing with context, a lack of direction for debugging, and the absence of self-knowledge useful for learning by generalization and analogy. Our next step is to define an annotation vocabulary consisting of formal comments regarding the plans, bugs, and interrelationships of the basic productions. We show how these annotations support more efficent execution of the skill, debugging of difficulties and self-improvement.

Annotated production systems represent a marriage of the comment-based approach to debugging developed by Sussman [73] and Goldstein [74] with the procedural architecture of production systems [Newell & Simon 72]. Goldstein and Sussman were not concerned with the psychological validity of their debugging models. Production systems have typically not been concerned with modelling learning. The marriage of productions and annotations holds out the possibility of an improved modelling capability. Davis [76] develops a related theory of mcla-knowlcdge for production systems to guide the knowledge acquisition process for large knowledge-based programs which we discuss in section 9.

Our work on annotated production systems is based on a series of experiments with a flight simulator implemented in Lisp by the authors. The production systems and annotated production systems discussed here run in conjunction with this simulator. The debugging and learning modules have been hand-simulated, but not implemented.

Unlike the traditional Carnegie Mellon experiment in which particular individuals are modelled, our experiments arc concerned with generic modelling; that is, they are concerned with modelling typical skill states of student pilots, as judged by the extensive literature on flight instruction (Langewiesche [44] remains the classic text) and the experience of the first author and his spouse in learning to fly.

In the next section, we characterize instrument flying to indicate why we have chosen it as our experimental skill. Section 3 describes the strengths of a production system model for this skill. Section 4 analyze! its weaknesses. Sections 5 and 6 define an annotated production model and indicate its utility for more efficient skill execution and for self-debugging. Section 7 describes the design of a heuristic learning program for Annotated Production Systems.

2. Attitude Instrument Flying

Our experimental focus has been on attitude instrument flying wherein the goal is to maintain steady climbs, turns, descents or level flight. It is the basic constituent skill of instrument flying.

Flying, as a whole, is an appealing domain for studying skill acquisition because: (1) there is extensive literature on flight instruction; (2) it is an important skill whose improper employment risks lives; (3) it is an adult (as opposed to infant) learning experience and hence instrospcclive evidence is available; (4) a useful application of an improved analysis of the learning process for this skill is the design of a computer instructor for flight simulators; (5) it is representative of an important class of real-time control skills such as sailing, driving.

Instrument flying, in particular, has a constrained set of perceptual inputs — the instruments -- and a restricted set of actions — the controls. Attitude instrument flying, while a sub-skill of instrument flying, is still sufficiently rich to be an interesting modelling problem.

Let us consider a few of the problems associated with flying a plane, in order to define the characteristics of a generic model for flight skill. The first observation is that flying involves responses to the external forces of the environment — gravity, air movement, and other factors. This requires instruments to monitor the plane'* state and controls for changing that state. Hence, for each goal, a mapping from measurement to control is required.

However, such a mapping cannot be from one instrument to one control, since the higher order effects would be neglected. Thus, using only the value of the vertical velocity indicator to manipulate the elevators while trying to achieve level flight will not always succeed. A better model would take into account vertical acceleration. Without a sense of the second derivative, the pilot will over or under control the aircraft.

The mapping must be context sensitive. A control response appropriate in normal situations may fail in special contexts. For example, under normal circumstances, the goal of straight and level flight can be achieved by sequentially attempting these goals, i.e. the pilot can concentrate on establishing the proper heading, and only when it is within tolerance, direct his attention at the altitude. The rationale for this is that in normal circumstances the two processes are virtually independent. However, if the aircraft is in a stall (i.e. the wing has lost lift), then the assumption of independence of the two subgoals is not valid and special measures must be taken in order to recover from this state. The wings must be leveled before the pitch of the plane is corrected. A representative set of flight contexts are:

- NORMAL FLIGHT
  - TAKEOFF
    - VISUAL FLIGHT TAKEOFF
      - SHORT FIELD TAKEOFF
      - SOFT FIELD TAKEOFF
  - LANDING
    - VISUAL FLIGHT LANDING
    - INSTRUMENT LANDING
  - CRUISE
    - STRAICHT AND LEVEL
    - CLIMB
    - DESCENT
    - TURN
- ABNORMAL FLIGHT RECOVERY
  - ENCINE FAILURE
  - INSTRUMENT FAILURE
  - STRUCTURAL FAILURE
  - NAVIGATION FAILURE
  - NON-NORMAL FLICHT CONDITION
    - COLLISION COURSE
    - STALL
    - SPIN
    - STABLE FLIGHT OUTSIDE TOLERANCES

Context sensitivity raises the issue of exceptions. A particular method may apply in all but a few situations. These exceptions should be explicitly accounted for if the skilled practioncr is to successfully anticipate them. Note that this is not the case of two competing heuristics, each equally applicable and each applying to roughly the same number of situations. Rather, we have the situation of one heuristic working almost everywhere and only a few exceptions need to be noted. For example, it is almost always the case that the ailerons are used to bank the airplane. In rare circumstances such as a spin, the rudder is used to level the wings. Such rare circumstances arc explicitly known as exceptions by pilots — indeed, much of flight training concentrates on the exceptions.

The flight world cannot be decomposed into orthogonal control dimensions. Instead, actions in one dimension effect other aspects of the flight of the aircraft. Thus, changing the bank of the aircraft by manipulating the ailerons will also cause a change in the pitch of the aircraft. There is no one-to-one mapping of the variables of the situation onto the set of controls. These interrelationships are in part the cause of the context sensitivity noted above. Some of the interrelated control effects are:

Rate of turn it controlled by both ailcront and rudder.
Kate *of climb it controlled by both throttle and clovatort.*
In landing t, rate of climb it affected by flaps and landing gear.
In tteep turns, rate of climb it affected by ailcront.

Finally, because this is a dynamic situation, time plays an important role. Hence, not only is the action which is chosen and applied important, but so is the rate at which that action is applied, and the timo period over which the corrections are applied. This introduces problems such as overcontrolling, a direct result of these dynamics.

3. A Production System Model for Attitude Instrument Flying

PONTIUS-0 is a production system for achieving straight and level flight that embodies a mapping from goals and measurements to controls. Below are a few representative productions without their annotation commentary. The patterns describe the goals of the productions; the actions observe instruments and manipulate controls.

```
(DEFINITION SftLl ;S«stra1ght. L-l«vtl.
 ;;To achieve straight and level flight, first achieve
 ;;level flight, and then straight flight.
  (GOAL: (AND (S FLIGHT) (L FLIGHT)))
  (ACTION:   (DO (ACHIEVE (L FLIGHT))
                 (ACHIEVE (S FLIGHT)))))

(DEFINITION  L-FLIGHT-1
 ;;To achieve level flight, keep the pitch of the plane at zero,
 ;;where the pitch 1s the angle of the nose with the horizon.
  (GOAL: (L FLIGHT))
  (ACTION:  (DO (ACHIEVE (NOTICE DELTA PITCH))
                (ACHIEVE (MAKE PITCH 0)))))

(DEFINITION  NOTICE-DELTA-PITCM-VIA-ARTIFICIAL-HORIZON-1
 ;;If the nose 1s down according to the artIflcal horizon,
 ;;then assert this fact m memory.  This 1s one among 4
 ;;mtthodi for noticing the pitch of the plane.
  (60AL: (NOTICE OELTA PITCH))
  ;;QUAI-VALUE Returns the sign of Its input.  Thus,
  ;;these productions are sensitive to the qualitative
  ;;va1ue of the Instruments.
  (ACTION:  (COND (IF ?(- (QUAL-VALUE ARTIFICIAL-HORIZON-PITCH) ♦)
                      .(■ (QUAL-VALUE (DELTA PITCH)) +))
                  (IF ?(• (QUAL-VALUE ARTIFICIAL-HORIZON-PITCH) -)
                      .(■ (QUAL-VALUE (OELTA PITCH)) -)))))

(DEFINITION  NOTICE-DELTA-PITCH-VIA-VERTICAL-VELOCITY-INDICATOR-1
 ;;If the plane 1s descending, the nose 1s down.
  (GOAL: (NOTICE DELTA PITCH))
  (ACTION:   (COND (IF ?(• (QUAL-VALUE WI) -) "THEN"
                       .(• (QUAL-VALUE (DELTA PITCH)) -))
                   (IF ?(■ (QUAL-VALUE WI) ♦) "THEN"
                       .(• (QUAL-VALUE (DELTA PITCH)) ♦)))))

(DEFINITION  CONTROL-PITCH-VIA-ELEVATORS-1
 ;;If the nose 1s down, pull up on the elevators.  Another
 ;;contro1 for pitch manipulates the throttle.
 (GOAL: (MAKE PITCH 0))

 ;;DELTA-ELEVATORS 1s a primitive control actions.
 (ACTION:  (DELTA-ELEVATORS (MINUS ?(DELTA PITCH)))))
```

The "?" preceding a form indicates that the form is a predicate whose truth value of T *or* NIL is computed by pattern-matching against the database. The "." indicates that the following form is to be asserted in the database, rather than being executed.

PONTIUS-0 has approximately SO rules for attitude instrument flying. A representative list for straight and level flight arc is given below where each title refers to one production by its goal.

```
PRODUCTIONS FOR ACHIEVING STRAIGHT AND LEVEL FLIGHT:
 SEQUENTIAL PLAN FOR STRAIGHT-FLIGHT AND LEVEL-FLIGHT
 COROUTINE PLAN FOR STRAIGHT-FLIGHT AND LEVEL-FLIGHT

PRODUCTIONS FOR ACHIEVING LEVEL-FLIGHT:
 ACHIEVE  L-FLIGHT
 NOTICE  DELTA  PITCH  VIA  ARTIFICIAL  HORIZON
 NOTICE  DELTA  PITCH  VIA  WI
 NOTICE  DELTA  PITCH  VIA  ALTIMETER
 NOTICE  DELTA  PITCH  VIA  AIRSPEEO
 ELIMINATE  DELTA  PITCH  WITH  ELEVATORS
 ELIMINATE  DELTA  PITCH  WITH  THROTTLE

PRODUCTIONS FOR ACHIEVING STRAIGHT-FLIGHT
 ACHIEVE  S-FLIGHT
 NOTICE  DELTA  BANK  VIA  ARTIFICIAL  HORIZON
 NOTICE  DELTA  BANK  VIA  TURN  COORDINATOR
 NOTICE  DELTA  BANK  VIA  DIRECTIONAL  GYRO
 NOTICE  OELTA  BANK  VIA  MAGNETIC  COMPASSS
 ELIMINATE  DELTA  BANK  WITH  AILERONS
 ELIMINATE  DELTA  BANK  WITH  RUDDER
```

These rules have a standard pattern/action form. The rules arc invoked in a depth-first method. That is, given a goal to achieve, the actions corresponding to that goal arc achieved in a depth-first manner by matching ACHIEVE patterns against COAL descriptions. If more than one production matches in a situation then the default order of calling is used. For example, there are several methods of noticing a change in pitch.

Production systems arc an appealing representation for the flight world for several reasons. First, the knowledge of how to fly an aircraft can be represented to a first approximation as a sequence of independent "recognize-act" pairs. That is, it can be represented as a sequence of rules of the form: given some goal and some context, do this action to bring the state of the aircraft "closer" to the desired state. Production systems offer a convenient formalism for structuring and expressing that knowledge. Fig. 1 shows the performance of PONTIUS-0 in maintaining a shallow bank.

Second, it is important in a dynamic system to detect and deal with a large number of independent states. One must be able to react quickly to small changes. Production systems faciliate such a detection and reaction process. This is since any rule could possibly be the next to be selected, depending only on the state of the data base at the end of the current cycle. Thus, each rule can be viewed •a a demon awaiting the occurrence of a specific state.

4. Limitations of Production Systems

The assumption that all rules are independent carries with it the additional assumption that all rules *arc* equally likely to bo used at any stage of operation. In this case, since the rules are sensitive to context, such an assumption is not valid. Specifically, some contexts are much more common and likely than others. Titus the rules are weighted in a certain sense and a formalism which accounts for this weighting would improve the performance of the model. Similarly, the fact that exceptions to situations exist should also be accounted for. Once again, a weighting factor is involved as the exceptions arc much rarer than the normal situations. Since we arc dealing with a dynamic real time system, performance is crucially linked to reaction time. As a result, it is important for all possible efficiency considerations to be used. This is why the weighting factors must be taken into account.

There are interactions in the flight world. Thus, there should be some capability for communication between rules. In production systems, there i6 only a limited communication between actions since such communication must take place via the short term memory data base. In a flight situation, this is inappropriate. For example, changing the bank via the ailerons can cause a change in the pitch. One way to communicate this fact is to actually manipulate the ailerons and let the system notice the change in pitch. But there should be an easier and more certain method of communication, for example, to alert the system that the manipulation of the ailerons may have effects on the state of the pitch.

The assumption of the independence of actions is not always valid cither. In normal situations, this is the case. However, there arc situations where parallel processes, or other complex procedures should be used. Production systems, however, arc at their best when actions are independent, and are not well-suited to coordinate processes. Fig. 2 illustrates an unsuccessful steep turn — the nose down pitch caused by the steep turn has not been corrected rapidly enough by PONTIUS, which is executing a sequential plan for straight and level flight.

The bug of paying undivided attention to the current goal and ignoring other subgoaU is a standard error of the student pilot. Much of instrument flying is devoted to establishing the proper "scanning pattern". The result of erroneous scan in the case of the steep turn shown in fig. 2 — entering a dive — is a common behavior of instrument students. PONTIUS exhibits many instances of such standard errors, and it is in this sense a generic model. Fig. 3 will show PONTIUS correcting this underlying bug by establishing a proper scanning pattern or "coroutine plan".

Production systems have a restricted syntax which means that the action side of the rules is restricted to a conceptually simple operation on the data base. This makes it difficult to include complex actions like coroutines or time sharing processes.

Another common problem associated with production systems is the "implicit context problem". This is the fact that the rule base has a total ordering associated with it and the position of the rule in this ordering becomes an important factor. Thus, since a rule ordinarily won't be called unless tho rules preceding it in the total order have failed, there arc in essence extra conditions on the application of the rule. This may affect the performance of the system.

These are some of the problems associated with using production systems as a representation for the performance component of these models of control skills. As a consequence of these problems, modifications were made to the production system formalism in order to improve the performance of the system. This resulted in the formation of annotated production systems.

S. Annotated Production Systems

Annotated production systems extend ordinary production systems by adding commentary to the productions. This allows one to represent second order knowledge explicitly and therefore to use this knowledge to handle some of the problems mentioned in the previous section. These annotations include caveats, rationales, plans and control information.

The annotated version of the production for straight and level flight shown earlier is:

```
(DEFINITION S&L1
  (GOAL:  (AND ($ FLIGHT) (L FLIGHT)))
  (ACTION:   (DO (ACHIEVE (L FLIGHT)) (ACHIEVE ($ FLIGHT))))
  (PLAN:     (SEQUENTIAL-PLAN :ACTION))
  (RATIONALE:(IF ?(= STATE NORMAL) "THEN"
             .(INDEPENDENT (ACHIEVE ($ FLIGHT))
                           (ACHIEVE (L FLIGHT)))))
  (CAVEAT:  (BUG SUB-GOAL-FIXATION)          '
          (IF ?(= MANEUVER STALL) "THEN"
            .(NOT (INDEPENDENT (ACHIEVE ($ FLIGHT))
                               (ACHIEVE (L FLIGHT))))
            (SWITCH-PACKET STALL-RECOVERY))
          (IF ?(= MANEUVER SPIRAL-DIVE) "THEN"
            .(PRECEDES (ACHIEVE ($ FLIGHT))
                       (ACHIEVE (L FLIGHT)))
            (PUSH-PACKET SPIRAL-DIVE-RECOVERY))))
```

We have noted that in the current domain, the rules do not all have equal weight in terms of range of applicability, or likelihood of applicability. Thus, rather than creating a rule for each combination of goal and context, we employ caveats to account for multiple contexts affecting a goal. Hence, the normal context will have a production associated with it. These caveats describe the relationship of the goal of the production to various "non-normal" contexts. They may simply point out when assumptions implicit in the form of the production are invalid, as is the case in the second caveat; or they may provide explicit information about the planning necessary to achieve the goal in the non-normal context, as is the case in the third caveat. Many of the interrelationships between actions can also be represented by the caveats. These can serve as warnings about possible effects of the action part of the production, such as the first caveat which warns of subgoal fixation.

The "implicit context problem" is handled by adding second order knowledge to the system. Thus, the CONTROL comment of a production contains information regarding the use of a production in cases where more than one such production matches the current goal. For example, there are four productions to notice a change in pitch. These involve using the artificial horizon, the vertical velocity indicator, the altimeter and the airspeed. Information can be added to the productions to state that one of these methods is the primary method, that others should be used to verify the validity of the primary production, and still others should be used as backup in case the primary method is known to be inoperative. This is exemplified by the production for level flight.

```
(DEFINITION L-FLIGHT-1
  (GOAL: (L FLIGHT))
  (ACTION:  (DO (ACHIEVE (NOTICE DELTA PITCH))
                (ACHIEVE (MAKE PITCH 0))))
  (CONTROL: .(= (PRIMARY-METHODS (NOTICE DELTA PITCH))
                (FIND M "SUCH-THAT" (= :M :METHOD (VIA AH))))
            .(= (CHECK-METHODS (NOTICE DELTA PITCH))
                (- (METHODS (NOTICE DELTA PITCH))
                   (PRIMARY-METHODS (NOTICE DELTA PITCH))))
            .(= (BACKUP-METHODS (NOTICE DELTA PITCH))
                (CHECK-METHODS (NOTICE DELTA PITCH)))))
```

One of the advantages of a production system is that the structuring of information as a collection of rules allows the system to generate explanations of its actions fairly easily. By making explicit more of the knowledge embedded in the system, we can enhance the explanation facilities. This is exemplified by the rationale comments, which describe the overall plan justifying the nature of the action. As well, rationales for the use of particular productions are attached to the productions themselves, to that explanations are further aided. For example, if the system was questioned about why it was attempting to achieve straight flight, it could respond that it was attempting to achieve the higher level goal of straight and level flight. If it was further questioned about why it was doing this in the particular method chosen, in this case a sequential plan, the system could use the rationale to explain that in

a normal situation, the two subgoals are essentially independent.

Such a facility for explanation, and in particular the rationales, also aids the system in debugging its performance, by pinpointing the likely source of error. To further this debugging process, models of plans and general bug types are stored with the system. These models can then serve to provide a context for debugging and repair. The plan associated with each production is attached to the production. The caveats may also contain pointers to new plan types which may be used in case of failure.

Specifically, we have the following plan taxonomy, with indentation indicating successive specialization.

PLANS
- CONJUNCTIVE
  - INDEPENDENT
    - PARALLEL
    - SEQUENTIAL
  - DEPENDENT
    - ORDERED
    - COROUTINE
    - GLOBAL
- CAUSAL
  - CONTROL
    - OPEN
    - FEEDBACK
  - MEASUREMENT
    - DIRECT
    - INDIRECT

Associated with this taxonomy of plans is a taxonomy of bugs. For example, a sequential plan in a real-time situation is susceptible to the bug that while one goal is being pursued, the other gets out of hand. We view debugging as a transformation process between plans. Hence, debugging a sequential plan might mean to employ the alternative of a coroutine plan in which processing time is shared between subgoals. To illustrate this, consider the following situation. We are attempting to turn the aircraft while maintaining level flight. In the rate of turn desired is small, the two goals can be considered independent and a sequential plan is appropriate. This was the plan cmployod in the successful maneuver of fig. 1. However, if the rate of turn desired is large, then the two goals arc no longer independent and there is an unexpected dependency. So we have a linear plan bug. This was illustrated in fig. 2. It is repaired by changing the plan to a coroutine plan, in which attention is tirncshared between the subgoals. Fig. 3 illustrates PONTIUS successfully flying a steep turn when told to employ a coroutine plan. Currently, transformations between plans can be requested of PONTIUS and the appropriate modifications made by accessing annotations. Automatic debugging is not yet implemented.

By attaching these annotations to the productions, the performance of the system is greatly enhanced. Among the effects are: an explanation capability, automation of debugging, efficient structuring of the procedural knowledge, and the use of complex processes such as parallel processes or timesharing processes. Because we are dealing with a real time situation, performance efficiency becomes an important factor and annotated production systems show a large improvement in this dimension over ordinary production systems.

6. Interpretation of Annotated Productions

Using annotations, there are 3 ways in which the productions can be interpreted.

(1) Standard Interpretation: The simplest possible operation of the performance component of this system uses only the basic portion of the production rules in a standard pattern directed mode. In this mode the annotations are used only during debugging and serve to help explain the difficulty and possibly correct it.

(2) Directed Interpretation: An improvement over this mode of operation is to allow a more sophisticated capability for handling situations in which multiple productions match the current goal. This mode is governed by the search advice contained in the CONTROL annotation, as was illustrated by L-FLICHT-1. This control information specifies whether the search should be depth-first, breadth-first or some intermediate variety allowing for the possibility of suspended nodes. Such specification is accomplished, in part, by stating whether a method is "primary", for "checking" or for "backup". The selection criteria can cither be explicit predicates or can be deduced from other commentary.

(3) Careful Interpretation: A further improvement is to access the commentary in each production, before the production is executed. The commentary is used to verify the appropriateness of the production, its success, and the appropriate actions to take upon failure. Thus, if a annotated production states that it is applicable in the normal context but not in all contexts, this mode checks the context as a whole, and not just the state variables being accessed directly by the production, to check whether the normal state is in effect. Similarly, if the system notes that several strategics are available for the same goal, all arc tried and compared. If inconsistencies exist, then the rationales and caveats arc checked for an explanation.

7. Learning

Annotations can provide the data for a heuristic compiler capable of modifying the production system to achieve progressively improved levels of performance. We have not implemented such a compiler, but our plans for its design arc based on the following six techniques: (1) the creation of specialists, (2) the use of caveats, (3) the use of plans, (4) learning by generalization of the plan, (5) learning by analogy, and (6) efficiency considerations.

(1) Specialist Creation: the organization of productions with a common calling pattern into a specialist is one powerful technique. For example, standard execution consists of simple pattern directed calls. Alternatively, a specialist may be constructed to dynamically decide which productions from a set with a common goal should be applied, the order of application, whether confirmation is necessary, which should serve as backup upon failure, whether a coroutine search is required, etc. In directed interpretation, such decisions are made on the basis of explicit CONTROL advice, e.g. statements that some methods are primary, while others are intended for backup or verification. Specialist creation compiles this advice by creating a separate "specialist" production which then calls-by-name, in the desired order, the various productions mentioned in the control annotation. The original set of productions with a common calling pattern are erased from the global context and asserted only in the local context of the specialist. Only the specialist is asserted in the global context. Hence, this aspect of heuristic compilation represents the understanding of the interrelationships between pieces of procedural knowledge that have a common goal.

Note that such a choice is strongly motivated by efficiency considerations, due in part to the real time nature of the domain. One problem which could arise, however, is if the rule base is incremented. Then the specialist would not take note of this new rule and would have to be updated, a possibly costly and difficult job.

(2) Caveat Checking: another aspect of the heuristic compiler is deciding where to check for caveats. Careful interpretation checked at the local level of entry into the productions. An alternative is to move a caveat from a position inside a production, where it is accessed only when the system is considering execution of the production, to an entry check associated with goals higher up in the hierarchy (thereby triggered preventing its original production from even being considered).

The heuristic compiler may also notice that all (or many) productions with a common goal have the same caveat and decide to introduce a specialist for these productions which checks the caveat before considering any of them.

The caveat may bo serviced in two ways. It can be examined upon entry to the method. Alternatively, the caveat can be compiled into a demon which remains active for as long as the method is on the goal stack. In this latter case, the caveat is constantly monitored during the period during which the action of the method is being executed.

The system can be informed specifically of the kind of servicing desired for the caveat: for example, entry caveat, exit caveat, continuous caveat; or this can be deduced from the nature of the caveat's test.

(3) Plans: heuristic compilation can also involve a consideration of the consequences of different planning approaches — control plans, linear plans, ordered plans, coroutine plans, iterative plans. This is used to provide more determinism and direction in the organization of the system. For example, the use of plan characteristics to debug errors was illustrated in moving PONTIUS from a sequential to a parallel plan for a steep level turn. However, this was done manually. PONTIUS docs not yet diagnose these difficulties by itself.

(4) Generalization: another function of the heuristic compiler is generalization. An example of this is where a student has learned a packet of productions for level flight and is then told that to achieve climbing flight, it is only necessary to generalize these productions in such a way that the desired pitch is transformed from a constant (zero) to a variable. For example, L-FLICHT-1 can be transformed to CLIMB-FUCHT-1:

```
(DEFINITION L-FLI6HT-1
 (GOAL:(L FLIGHT))
 (ACTION: (DO (ACHIEVE (NOTICE DELTA PITCH))
              (ACHIEVE (MAKE PITCH 0)))))

(DEFINITION CLIMB-FLIGHT-1
 (GOAL:(CLIMBING FLIGHT TO 7ALTITUDE))
 (ACTION: (DO (ACHIEVE (NOTICE DELTA PITCH))
              (ACHIEVE (MAKE PITCH 7VARIABLE)))))
```

Alternatively, one could teach the system climbing flight as a separate primitive packet and let the heuristic compiler notice that the two packets have a common generalization. Then the two packets could be replaced with the common generalized version.

(5) Analogy: another process used to create new methods from old ones is "analogous reasoning". For example, the entire packet for straight flight might be constructed from the previously learned packet for level flight using the analogy:

```
  PITCH        --> BANK;
ALTITUDE    --> DIRECTION;
ELEVATORS   --> AILERONS;
FEET        --> DEGREES;
VERTICAL VELOCITY INDICATOR --> TURN COORDINATOR;
ALTIMETER   --> DIRECTIONAL 6YR0;
ALTIMETER   --> COMPASS.
```

This would have to be debugged, but it provides strong guidance in the initial program construction process. Using this mapping, S-FLICHT-1 can be created from L-FLICHT-1.

```
(DEFINITION S-FLIGHT-1
  (60AL: (S FLIGHT))
  (ACTION:  (DO (ACHIEVE (NOTICE DELTA BANK))
                (ACHIEVE (MAKE BANK 0))))
  (CONTROL:
    .(■ (PRIMARY-METHODS (NOTICE DELTA BANK))
        (FIND M -SUCH-THAT" (• :M :METHOD (VIA AH))))
    .(■ (CHECK-METHODS (NOTICE DELTA BANK))
        (- (METHODS (NOTICE DELTA BANK))
           (PRIMARY-METHODS (NOTICE DELTA BANK))))
    .(■ (BACKUP-METHODS (NOTICE OELTA BANK))
        (CHECK-METHODS (NOTICE DELTA BANK))))
  (CAVEAT:
    (BUG METHOD-FIXATION
         (GOAL: (NOTICE DELTA BANK))
         (METHOD: PRIMARY))))
```

(6) Efficiency: heuristic compilation techniques related to efficiency include finding tubgoals which can be accomplished by a •ingle action. For example, different goals may require the same information. The naive approach would be for each of these subgoals to notice the required state variable independently. The heuristic compiler would instead use memory to record the result. Then the second goal could aave time by accessing memory.

8. Research Plans

(1) Our current goal is to continue the experimental investigation of annotated productions as a model of generic flight skill. We plan to implement a heuristic learning program that can successively modify an inital APS model in response to flight experience obtained from the behavior of the model in controlling the simulator and coaching based on the standard instructional sequence found in flight textbooks. Success will be judged by the extent to which the APS evolves into a competent pilot, exhibiting and correcting typical piloting bugs.

(2) The next goal will be to model individual pilots. We plan several experiments along this line directed towards protocol analysis of student pilots flying our Lisp simulator and the Orly simulator developed by Feurzcig and Lukas [1975]. Our hypotheses is that it will be possible to evolve an APS model for individual students that predicts common errors.

(3) The third step will be to automate this protocol analysis, using the techniques of overlay modelling developed in [Carr and Goldstein 77]. These techniques constitute a general methodology for generating information processing models, if a modular and comprehensible expert program for the domain is provided. PONTIUS will provide this required expertise.

(4) Our ultimate goal is the design of a Computer Coach for flight simulators that analyzes a student's flying and coaches him on the underlying control skills. The theory of computer coaches is developed in [Goldstein 77]. If APS provide the necessary model of expertise, then we believe that the rule-based tutoring theory developed in [Goldstein 77] will lead to computer coaches that can significantly improve the effectiveness of flight simulator training for students and professional pilots.

9. Meta-Knowlcdge for Large Knowledge-Based Systems

Annotations are a kind of mota-knowlcdgc. Davis [76] develops mcta-rulcs and other types of mcta-lcvcl knowledge for use in association with the MYCIN system [Shortliffc 74]. In particular, this meta-knowlcdgc is used to aid the explanation by the program of its actions, to automate the addition of new knowledge, and to direct the use of the object level knowledge. The rneta-rulcs which accomplish the latter arc similar to our specialists.

However, we believe that further aspects of the annotated production system would be appropriate for the medical domain of MYCIN which are not included in Davis* TEIRKSIAS program. For example, the use of rationales could improve the explanation facilities. Currently, MYCIN/TEIRESIAS uses the action of each rule as a basic unit of explanation. While this does explain the actions of the program, it docs not consider the underlying justification for those actions. Rationale slots could be used to *carry* such justifications, for example, the reason that medical researchers believe the rule to be valid. This would be critical if MYCIN is ever to be part of a computer coach for medical students.

A second possibility is in the use of plans. Doctors, in approaching some problems, create and use plans. For example, drug therapy, the domain of MYCIN, is usually only a step in the overall treatment of the patient. MYCIN currently does not have a representation for explicit plans: annotations provide a natural extension to production systems to make explicit planning knowledge.

A third possibility is to group less frequently used productions for a given goal into caveats associated with their more frequently employed brethren. The caveat would be triggered by some warning in the global database. For example, it might be appropriate to separate diagnostic rules appropriate for an emergency from standard diagnostic procedures by means of caveats. Greater efficiency and modularity is obtained by thereby reducing the size of the currently applicable knowledge base.

10. Conclusions

In the seminal work on production systems by Nowcll and Simon, the task if explicitly limited to modelling an individual engaged in a non-learning situation. Hence, meta-knowlcdgc in the form of commentary was not a part of the production model. However, as we have demonstrated for the flight domain, meta-knowlcdgc is critical

when the problem of an individual improving his skill is addressed. This paper has introduced a formal vocabulary for some of this knowledge. We believe these annotations constitute a small stop towards a theory of self-knowledge which may well be the essential ingredient to the design of large knowledge-based systems capable of self-improvement, explanation, and sufficient efficiency for real-time processing.

11. References

Carr, B. and I. P. Coldstcin. 1977. Overlays; A Theory of Modelling for Computor Aided Instruction. M1T-AI Memo 406.

Davis, R. 1976. Applications of Mcta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases. SAIL Memo 283.

Feurxeig, W. and C. Lukas. 1975. Higher Order Adaptive Training Systems, Bolt, Bcranck and Newman Proposal P76-1SD-14.

Coldstein, I. P. 1974. Understanding Simple Picture Programs. MIT-AI TR-294.

Coldstein, I. P. 1977. The Computer as Coach. MIT-AI Memo 389.

Langiewisch, W. 1944. Stick and Rudder. McCraw Hill, New York, republished 1972.

Newell, A. and H. Simon. 1972. Human Problem Solving. Prentice Hall, Englewood Cliffs, N. J.

Shortliffc, T. 1974. MYCIN — A Rule-based computer program for advising physicians regarding anitmicrobial therapy selection. SAIL Memo 251.

Sussman, C.J. 1973. A Computational Model of Skill Acquisition. MIT-AI TR 297.

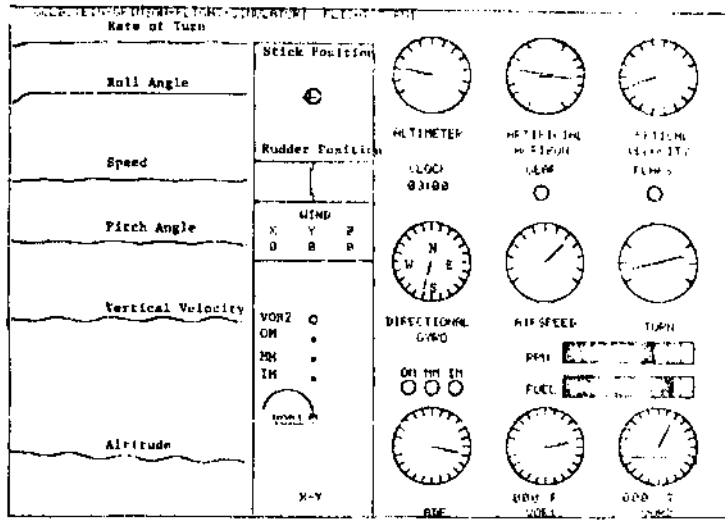

FIGURE 1

SUCCESSFUL SEQUENTIAL SHALLOW BANK

The instruments are being sampled *every* 5 seconds. First the productions for level flight are executed until the altitude is within the desired tolerance and then the productions for turning, again until the rate of turn is within tolerance.

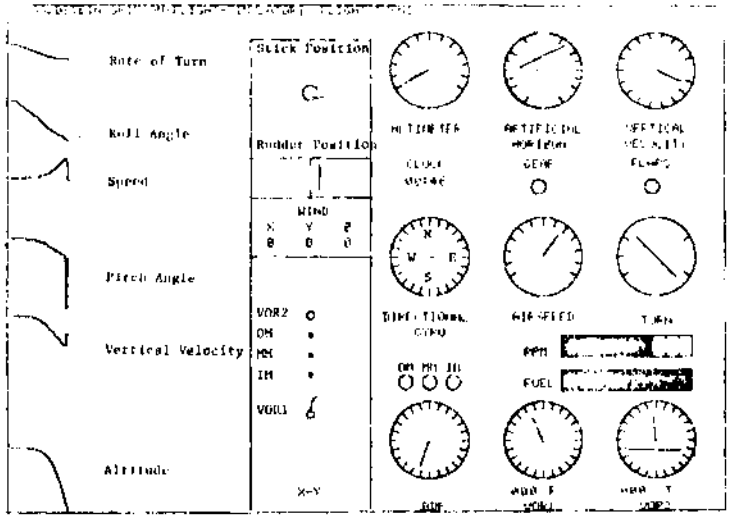

FIGURE 2

UNSUCCESSFUL SEQUENTIAL STEEP BANK

The instruments are being sampled *every* 5 seconds. First the productions for level flight are executed until the altitude is within the desired tolerance and then the productions for turning, again until the rate of turn is within tolerance. Unfortunately, the plane crashes before PONTIUS has established the desired rate of turn.

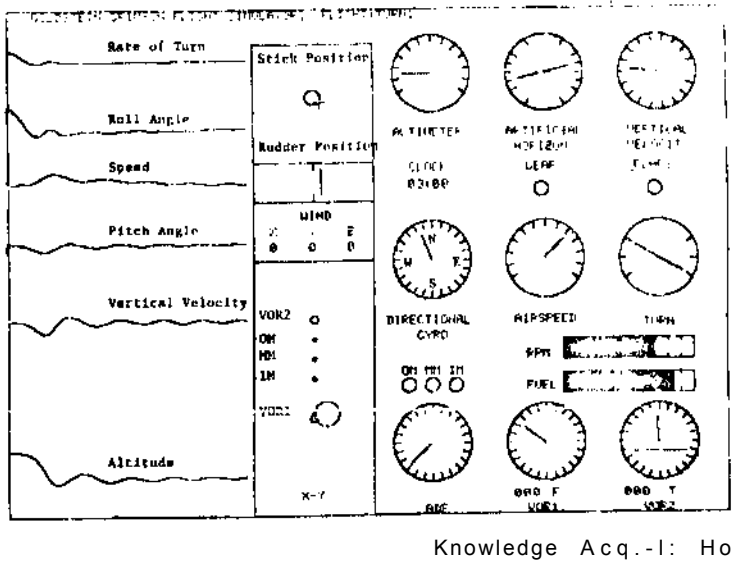

FIGURE 3

SUCCESSFUL PARALLEL STEEP BANK

The instruments are being sampled every 5 seconds. Attention is divided between the productions for level flight and productions for turning.