A Synthetic Architecture for Action and Learning

Charles Earl and R. James Firby
Artificial Intelligence Laboratory
Computer Science Department
The University of Chicago
Chicago, IL 60637
(312) 702-6209
earl@cs.uchicago.edu, firby@cs.uchicago.edu

Abstract

Our project concerns the development of a system which integrates situation driven execution with constructivist learning. We begin with reactive planning as embodied in the RAP planning and execution architecture [Firby, 1989]. We describe how a system originally designed for unstructured constructivist learning, Drescher's Schema learning mechanism [Drescher, 1991], can be modified to support similar goal directed reactive behavior. In particular, we have adapted the system to pursue explicitly defined goals and developed a set of macros for specifying composite schema and actions. Using the Truckworld simulation system [Firby, 1989] as a testbed we were able to verify that the modified Schema mechanism is able to perform much like the RAP system. Further, the mechanism learned information relevant to the systems explicit goals.

1 Introduction

This paper presents initial results in integrating situation driven execution with a computational model of constructivist learning. In recent years, researchers in the area of planning have realized the necessity of incorporating an ability to react to rapidly changing, uncertain circumstances into their systems. Unfortunately, learning has received scant attention in such research even though constructivists have long argued that intelligent agents construct an understanding of the world *concurrent* with their activity in it [Papert, 1980].

The goal of this research is to develop an architecture for situation driven execution with the intrinsic property of continually learning about important features of the environment. It will allow an agent to react intelligently as unforeseen circumstances arise in the environment and will also allow the agent to notice important features and learn causalities which will enable the agent to improve its performance. Our starting point is to merge ideas from the RAP system for adaptive plan execution in a dynamic world [Firby, 1989; Firby, 1987; Firby, 1990] with the schema mechanism [Drescher, 1991] proposed for unstructured constructivist learning.

In particular, we are adapting Drescher's schema mechanism to suit the needs of our reactive execution system. Our initial approach is to implement the schema mechanism, incorporate into it those gross features that make the RAP system work well, and then explore the detailed differences that don't carry over from one system to the other. This approach is motivated by the compelling parallels between the RAP system and the schema mechanism and the belief that learning should be an intrinsic part of all interaction with the world and

.

1.1 System Overview

We begin with the abstract two part architecture conceived with the RAP system and consisting of a planner and a reactive execution system [Hanks and Firby, 1990]. The reactive execution system embodies a significant level of intrinsic competence and is capable of achieving a wide range of goals in its own right. Its purpose is to hide much of the uncertainty and changing detail of the world from the planner by adapting actions quickly and appropriately as goals are pursued. The planner's job is to construct plans for novel goals using subgoals the reactive executor knows how to achieve and to look into the future and prevent the executor from making short-sighted mistakes with serious consequences.[1]

The system described in this paper takes the place of the RAP execution system in such an architecture. The RAP system works well as a reactive plan executor but it lacks mechanisms for learning the effects of actions and for compiling new methods. The schema mechanism, on the other hand, learns action effects and abstract actions (*i.e.*, compiles methods) as an intrinsic part of the way it works. Our goal is to create a system that maintains the explicit goal achievement orientation of the RAP system while including the learning characteristics of the schema mechanism. Like the schema mechanism, our system will not use an explicit training phase to learn about the world, learning will be an intrinsic property of taking action. Like the RAP system, our executor will start with a large number of predefined methods so that it will immediately be able to pursue a multitude of complex goals. However, these initial methods will be

[1]The RAP system, and any realistic control architecture, must also deal with the problems of interfacing to an underlying real-time control system [Firby, 1992; Firby and Swain, 1991]. We ignore those problems for the time being.

open to revision and expansion via the learning mechanism and new causal knowledge learned about actions will be available to the planner for future use.

The similarities between RAPs and the schema mechanism are striking. Both use a hierarchy of known methods as the basic knowledge for achieving goals, both use very similar mechanisms for choosing between alternative methods for achieving the same goal, and both use measures of importance for shifting attention when unanticipated events occur. However, the RAP system stresses the knowledge and algorithm required for achieving a wide variety of goals while the schema mechanism stresses the knowledge and algorithm required to learn new causal knowledge during goal pursuit. As a result, the RAP system assumes goals arise externally and basic knowledge is available for achieving those goals while the schema mechanism assumes goals arise internally and a critical task is to explore actions that might be effective at achieving goals.

Our approach is to begin with the learning and action mechanisms from the basic schema mechanism and adapt them to: (1) allow the inclusion of initial methods for achieving goals; and (2) prefer goal pursuit over exploration.

2 The Schema Mechanism

The basic mechanism used in our system is based upon the constructivist learning mechanism proposed by Drescher [Drescher, 1991]. Drescher proposed an architecture designed to induce the causality of the world through an agent's repeated interactions with it. His concern was with how such an architecture could give rise to the kind of staged development elaborated by Piaget [Piaget and Inhelder, 1969]. We have modified Drescher's original implementation to run primarily in the pursuit of explicit goals (Drescher was principally concerned with what structures the system was capable of inducing in an exploration only mode). The mechanism defined by Drescher uses three principle data structures: schemas, actions, and items.

2.1 Basic Data Structures

2.1.1 Schemas

Schemas attempt to describe what happens when a given action is taken in a particular situation. A schema is a data structure which consists of three principle slots: a *context*, *action*, and *result*. The *context* slot expresses the conditions under which the schema is appropriate. A schema's *action* slot designates an action which can be executed in the state designated by the context slot. The *result* slot contains information on what changes in the world are likely to occur as a result of the action having been taken. Here, we will use the syntax **{context}schema{result}** to describe schemas. Note that the schema does not guarantee that its result will be obtained once the action is taken, it merely expresses the belief that it is likely.

2.1.2 Actions

Actions are commands which allow the mechanism to effect state changes in itself or the world. *Primitive actions* are analogous to simple agent reflexes, they are atomic units of activity. A *composite action* designates a set of schema, which when activated, are expected to result in a particular state of the world.

Associated with each composite action is an object called an *action controller*. The action controller supervises execution of the schemas listed in the action's *components* slot. When a composite action is created a *broadcast* is performed on the item's goal: a search is performed to determine the chains of schema which can accomplish the goal. This is analogous to a depth limited goal regression. The schemas which can accomplish the action's goal are then added to the controller's components slot.

2.1.3 Items

An *item* is a binary valued descriptor of state: an item just indicates whether a condition holds or not. Items have two slots, *relevance* and *primitive value* which indicate how generally relevant the item is to achieving an explicit goal or how important it is that a particular item state be maintained respectively.

2.2 The Schema Design Language

Our implementation of the schema mechanism includes a language for defining initial actions, schema, and items. A macro *defprimitive* allows a user to simultaneously specify simple schema and actions. These schema and actions correspond roughly to atomic actions. A *defitem* construct allows the user to define binary valued states items along with methods for obtaining their value from the agent's simulation environment.

The *defroutine* macro allows the user to specify the context and result items of a schema, and also allows the user to designate the components for the associated composite action. Hence, the user is able to define complex sequences of actions which allow the agent to act effectively in a complex world.

2.3 Schema Selection

The schema mechanism decides what to do next by letting schemas compete amongst each other for the opportunity to be run. A schema asserts its importance (i.e. need to be activated) via its activation level. A schema's activation level is computed as a weighted sum of values which express the schema's relevance to pursuing agent goals and its relevance to exploring interesting features of the world. The activation function is designed to emphasize goal pursuit over exploration.

A schema's relevance to the current goals of the agent is expressed by the values of its *goal-pursuit*, *subgoal-pursuit*, and *primitive value* slots. The synthetic architecture we are building receives goals that are generated by an external source. Upon selection of a new goal, the mechanism determines which of the valid schema contain that goal in their result slot and sets the value of their goal-pursuit slot to a value which expresses the degree to which the goal should be pursued.

The *subgoal-pursuit* slot enables the schema mechanism to include the fact that a schema has been indicated as being part of a composite action chain. When a composite action attempts to run a schema as part of its chain, it sets the subgoal-pursuit flag of the schema. The activation level of the schema will thus be incremented by an amount which reflects the mechanism's preference for pursuing schema which appear in composite action chains.

The *primitive value* slot indicates how relevant the schema is to maintaining conditions which should always hold.

The values of two other slots allow the mechanism to engage in exploratory behavior if no goals are present. The *habituation* parameter defines the rate at which the schema's activation increases on successive activations. This allows the agent to explore further the results of the last action taken. The *hysteresis-onset* and *hysteresis* slots determine when and how rapidly the activation of a schema will fall off after successive activations. These parameters prevent the agent from exploring the effects of one schema to the exclusion of others.

At each tick of the system clock, each schema computes its activation by summing values in the primitive value, goal-pursuit, and subgoal-pursuit slots with values computed for habituation and hysteresis. The control mechanism then selects the schema to run by choosing randomly from among the highest valued schemas.

Once an action controller has selected a schema to recommend for activation, it monitors the result of schema execution to determine whether the schema has succeeded. If the schema fails, it attempts to execute the schema for a fixed set of tries, and then gives up, proceeding to the next indicated schema on the queue. If the goal fails to obtain once the queue has been exhausted, the controller attempts to rerun itself from the beginning for a set number of tries and then gives up completely. The schema which invoked the action then notes that it did not succeed.

2.4 Generating New Schemas: Marginal Attribution

The central aspect of learning in the schema mechanism is noticing correlations between action (i.e. running a particular schema) and observable state changes (i.e. transitions in an item's state). Drescher refers to this process as *marginal attribution*.

Each schema includes an *extended result* structure which maintains two statistics for each existing item:

1. `Positive-transition correlation` This is the ratio of the probability that the item turns on when the schema is activated to the probability that the item turns off when the schema is not active.
2. `Negative-transition correlation` This ratio is analogous to the positive transition ratio, but with respect to the probability of the item being turned off.

During update, the schema uses the result of the last action to recalculate these values. If a schema detects that the item is more likely to be turned on (or off) after it has executed than when it has not, that item is deemed relevant and a new schema is created which list the item (or its negation) as a result.

When the mechanism determines the likely result of a schema's action, it then attempts to discover the conditions which allow it to complete successfully. To do this, each schema maintains an *extended context* structure which records for each item the ratio of probability of success when the item is on to the probability of success when the item is off. When one of the probabilities is significantly larger than the other, a new schema is generated with that item (or its negation, depending on which probability was larger) included in the context. The mechanism then adds items to its context using the same procedure, thus enabling it to learn conjunctive preconditions.

In practice, performing an update for each schema on statistics measured for each item severely degrades the performance of our system. We therefore enforce a policy in which only those schema whose activation level exceeds a threshold participate in the update process and restrict the items to those having made an `on -> off` or `off -> on` state transition within a preset window of cycles.

2.4.1 Creating Composite Actions

Each time that a spin off schema is created, the mechanism also determines whether the result of that action is unique. If it is, the mechanism creates a new action which lists the result as its goal. The mechanism then attempts to initialize the action controller for the new schema by determining those schemas which chain to the goal of the action.

3 A Situated Learning Architecture

The schema mechanism selects which schema to execute by letting schema compete on the basis of activation. In our system, a planner specifies goals for the action and learning mechanism in the same way as the RAP execution system . However, rather than spawning a task, such a goal increases the activation energy of schema listing that goal in their result slots (i.e., the goal relevance of the item is incremented). As stated before, this amounts to having the mechanism prefer explicit goal pursuit rather than the exploration based pursuit detailed by Drescher.

In addition, our schema definition language allows the creation of high level schema and actions prior to execution. Detailed plans of action that an agent is likely to need to behave effectively in its environment can be defined in advance.

Consider an agent which must use a truck to deliver bricks from a brick production sight to a construction area. A RAP for such a circumstance might specify that the bricks should first be loaded onto the truck, that the truck should pass a number of landmarks to get to the construction sight, and once there, that the agent should unload the bricks from the truck, and then return to the loading area to await further instructions. The RAP for accomplishing this goal could be be decomposed into a number of methods, for example, different plans for loading different kinds of objects.

In our hybrid system, this same information is represented as schemas and actions for loading the truck, moving past a series of landmarks to the construction sight, and then unloading the bricks once there.

After a number of deliveries, however, the schema mechanism is able to develop new schema which encapsulate information not specified in advance. In the delivery example, suppose that agent noticed that the truck was frequently stopped and its cargo seized along a certain portion of the route, say between the right turn at 3rd street and the left turn at 4th street. After a number of such incidents, it should be possible for the underlying schema mechanism to associate this portion of the route with an unwanted result and given a choice in the sequence of goals suggested by the high-level planner, prefer an alternate route.

Further, after a number of runs, it should be possible for the mechanism to compile a schema/action which improves the agents performance on the task. Initially, the planner specifies goals at fine levels of granularity. The low level action selection and learning mechanism begins with knowledge about how to accomplish primitive actions (in the example cited, lifting a brick say) and some higher level schema that may be considered to be generalizable across a range of likely situations (refueling the truck at a gas station for example). As the agent is presented with tasks to accomplish in the world, the underlying learning system is able to learn low level operators that correspond to the goal sequences previously "fed" to it by the planner.

3.1 Specifying Goals

The current implementation is designed to carry out goals specified by some high level reasoning component. As we do not yet have a working model of such a component, we let the user specify goal sequences for the agent to carry out via structures called *scripts*. A script specifies the way in which a set of goals is to be achieved and provides a mechanism for monitoring the agent's progress towards accomplishing those goals.

At each tick of the clock, the action selection mechanism is presented with the current high-level goal as determined by the script. The goal then influences the schema selection process and ultimately the selection of actions. In the full implementation, the exchange between the schema mechanism and goal generator will be bidirectional: whether the low level mechanism possesses a schema which can reliably accomplish the specified goal determines whether the goal needs to be further expanded into a set of context specific tasks.

4 Results

Our initial results are taken from the agent's interaction in a a simulated delivery environment called Truckworld [Firby, 1989]. The agent controls a truck capable of performing delivery tasks. The truck possesses two mechanical arms and two cargo bays. The simulated world consists of a set of locations connected together by roads and the truck can move along the roads from location to location. At each location, the truck may encounter a number items which it can manipulate in a number of ways. Principally these include: rocks, which can be grasped by either of two mechanical arms residing on the truck; fuel drums, which can also be grasped, and then used to refuel the truck; a gun, which can be mounted on the truck and loaded with ammo; and, enemy units, which seek to capture the truck.

Uncertainty takes a number of forms in the world. Enemy troops can be encountered without warning at any location in the world with the exception of the agent's home base and the fuel depot. Rocks are subject to disappearance in a probabilistic fashion (simulating the uncontroller actions of other agents). Arms occasionally drop objects and the degree of fuel consumption also varies.

4.1 An Initial Experiment

Typical tasks in the truckworld environment involve delivery of objects to specified locations. The experiment used in this investigation involves the truck agent in a relatively mundane task. It is given a script which instructs it to try moving to the east, grab a rock, then return to the original location, refuel, and then continue the loop.

Presented with a high-level goal from a script, the agent relies on roughly 300 RAP-like schema, items, and actions to define its initial information about the world. Briefly, these fall into categories of: (1) truck movement, or schemas and and actions which specify how to move about in the world; (2) arm movement, those schemas which specify how the arms can be moved to and from locations in the world, for example from the folded position to an object external to the truck; (3) protection primitives and routines, those which determine how the agent should behave when enemy units are sighted; (4) arm manipulation schemas and routines, which specify how objects can be held and moved by the arms.

The results from the experiments conducted so far indicate that our hybrid architecture is effective in combining a learning mechanism with a mechanism for situation specific goal directed activity without sacrificing the performance of either. That is, the agent effectively copes with the dynamic nature of its environment and learns as a result of that interaction. A critical issue remaining is whether or not the kind of structures that the schema mechanism learns can be of operational value to a reactive planner as we have assumed.

An example of how it copes with the unanticipated arrival of the enemy units illustrates this.

```
Active schema is <Generated schema AVOID-ENEMY>
Active schema is <Generated schema AVOID-ENEMY>
Active schema is <Primitive schema MOVE-TRUCK-EAST>
;Goal (:on <Simple item ROCK-GRABBED>) abandoned.
Active schema is <Primitive schema MOVE-TRUCK-SOUTH>
Active schema is <Generated schema GRAB-ROCK-1>
```

The agent has moved east and arrived at one of the quarry nodes in an attempt to grab a rock. After arriving at the quarry an enemy unit appears. As is indicated in the trace of activated schema, the AVOID-ENEMY schema is triggered by the prescence of enemy units. By the next cycle, the AVOID-ENEMY schema has passed activation onto one of the specified primitived schema, MOVE-TRUCK-EAST. There is no road to the east however, and the agent attempts escape to the south. This is successful and the agent resumes the script of foraging for rocks once it arrives at a safe location.

After several encounters with enemy units , the mechanism produces the following spin-off schema/action pairs:

```
Adding schema {}MOVE-TRUCK-WEST{ENEMY-IS-SENSED:OFF}
Adding schema {}MOVE-TRUCK-WEST{ROCK-SEEN:OFF}
```

The learning mechanism has been able to generalize several useful results from having avoided the enemy. The new schema, `{}MOVE-TRUCK-WEST{ENEMY-IS-SENSED:OFF}` is a specialization of the original `AVOID-ENEMY` schema which allows the mechanism to make some conclusions (perhaps premature) about the usefulness of the action of moving west. The second generated schema begins to hint at an even more interesting possibility, which is the generation of a map of feature related data.

5 Conclusions

The preliminary results from our work suggest two strong areas for further development: (1) definition and implementation of an effective goal generator; and (2) development of an architecture which integrates the goal generator and the synthetic system described here.

So far, we have ignored the issue of describing what the goal generating component should look like: the mechanism is simply given sequenced sets of goals to accomplish. The ideal planner would be able to generate goals for the schema mechanism not only in response to some higher level mission (e.g. collecting interesting rocks and returning them to the home base), but in response the the results of the low level component's success in accomplishing those goals. In other words, the planner would know when to intervene (i.e. specify new goals), when the lower level mechanism is having difficulty.

The goal generator should also be capable of making effective use of the knowledge gained by the low level mechanism. In other words, it must know how and when to alter its plans on the basis of new causal information.

This raises the issue of how to structure the goal generator so that it can effectively communicate with our existing situated learning architecture. The interface must allow the goal generator to determine what causal information has been learned by the low level mechanism. It must also allow it to extract the information needed to gauge the performance of the low level system.

In addition to investigating suitable goal generating mechanisms, we are expanding and modifying the existing set of actions, schemas, and items to deal with more complex tasks. As a result, our mechanism will have actions which allow it to focus attention on specific objects in its environment to restrict the set of items which must be monitored.

References

[Drescher, 1991] G. Drescher. *Made Up Minds: A Constructivist Approach to Artificial Intelligence.* MIT Press, 1991.

[Firby and Swain, 1991] R. J. Firby and M. J. Swain. Flexible task-specific control using active vision. In *Sensor Fusion IV: Control Paradigms and Data Structures*, Boston, MA, November 1991. SPIE.

[Firby, 1987] R. J. Firby. An investigation into reactive planning in complex domains. In *The Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, WA, 1987.

[Firby, 1989] R. J. Firby. Adaptive execution in complex dynamic worlds. Research Report 672, Yale University Computer Science Department, 1989.

[Firby, 1990] R. J. Firby. Planning, acting, and sensor fusion. In *Advances in Intelligent Robotic Systems*, Philadelphia, PA, November 1990. SPIE.

[Firby, 1992] R. J. Firby. Building symbolic primitives with continuous control routines. In *First International Conference on AI Planning Systems*, College Park, MD, June 1992.

[Hanks and Firby, 1990] S. Hanks and R. J. Firby. Issues and architectures for planning and execution. In *Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, CA, November 1990. DARPA.

[Papert, 1980] S. Papert. *Mindstorms.* Basic Books and Harvester, 1980.

[Piaget and Inhelder, 1969] J. Piaget and B. Inhelder. *The Psychology of the Child.* Basic Books, 1969.