A Critical Overview of the Schema Mechanism and Related Work

Jonathan Mugan

CS395T Intelligent Robotics
May 11, 2005

Introduction

The schema mechanism (Drescher 1991) enables a learning agent to construct a representation of how actions affect its environment. The schema mechanism is a an example of a *constructivist* AI system. In such a system, the learning agent begins only with very basic knowledge of its sensors and effectors and through interaction with its environment incrementally builds increasingly sophisticated behaviors. Constructivism traces its roots back the the developmental psychologist Jean Piaget (1952; 1954) who postulated a theory of development in which intelligence is built up in stages, with each stage building on the previous one. The schema mechanism is designed to serve as both a test of Piaget's theory, and as a possible framework for constructing artificial intelligence.

This paper will provide a detailed explanation of the schema mechanism from the perspective of an implementation on a serial computer. The strengths and weaknesses of the schema mechanism as a potential robot learning and control mechanism will be discussed, and related work on the schema mechanism will be surveyed. Finally, some directions for future work will be explored.

The Schema

The central organizing structure of the schema mechanism is the *schema*. A schema is a triple that consists of a *context*, *action*, and *result*. A schema is not a production rule, it does not say what action should be taken in a particular situation, but rather it states what would happen, with some probability, if a particular action were taken in a particular situation. An *action* is an event that can affect the state of the world. The context and result of a schema consist of state elements called *items*. Items are state elements that can take the values *On*, when true, *Off* when false, or *Unknown*. A context will contain one or more items designated *On* or *Off*, and a result will generally contain only one item except in the special case which will be discussed later. A schema's context is *satisfied* if all of its context items have the specified values, and a schema is *activated* if its context is satisfied and its action is taken. A schemas is said to *succeed* if it is activated and its result item(s) take on their specified value after activation, and it is said to *fail* otherwise. Each schema also contains statistics for **every** item, not just its context and result items, in its *extended context* and *extended result*. These statistics are for creating new schemas. The reliability of a schema is number of times it is successful divided by the number of times it is activated, and is given by

$$\mathsf{Reliability} = \mathsf{count(success)}/\mathsf{count(activation)}$$

A schema is said to be *reliable* if its reliability is above a threshold. A schema is *valid* if it would succeed if activated.

With respect to notation, schemas will be represented in the form $\langle \mathsf{ab}\neg\mathsf{c}|\mathsf{d}|\mathsf{fg}\rangle$ where a, b, and c are all context items, the action is d, and the result items are f and g. All items are considered to be *On* unless preceded by a $\neg$, thus in this case c would be *Off*. Items with names longer than one character will be separated by a &. Using the concrete, but high level example from (Chaput 2004), the schema

$$\langle \mathsf{InFrontOfDoor}|\mathsf{OpenDoor}|\mathsf{DoorOpen}\rangle \qquad (1)$$

would mean that if the learning agent were in front of the door, and it took the action of opening the door, then the door would be open. (Note that this example is at a much higher level of abstraction than the schema mechanism has been able to achieve, but it will be used throughout the discussion because of its intuitive appeal.)

Two Key Ideas

The schema mechanism specifically addresses two fundamental challenges in building artificially intelligent systems. The first is the challenge of *empirical learning*: given that the same action can have different effects in different situations, how can the learning agent learn which prerequisites are necessary to cause particular action to lead to a particular result? It is clearly intractable to try all combinations of prerequisites and actions to see which results follow; and a compounding problem is that results are sometimes brought about by causes other than those stemming from the learning agent's actions. The schema mechanism's solution to this difficulty is to use a method called *marginal attribution*, which breaks the problem into two parts. First, it finds results that follow actions if even with only slight reliability. Then, for each action/result pair, it incrementally looks for individual context items that increase the reliability of that result following that action.

The second challenge is *concept invention*: how can the learning agent add novel concepts to its ontology? The schema mechanism does this by finding schemas that are not *reliable* and are *locally consistent*. A schema is locally consistent if its success from very recent activations is a good predictor of subsequent success. Drescher gives no explicit formulation of what it means to be locally consistent, but a simple one would be to define it as the probability of a successful activation, given a successful activation in the last k time steps, is above a threshold. To add new concepts the schema mechanism creates a *synthetic item* for such schemas, and these synthetic items can then be in the context or result of any schema just like regular items. The synthetic item is *On* when its host schema would have a successful activation and thus represents the unknown state of the world that causes the host schema to be successful. If the synthetic item is *On*, then the host schema itself does not have to be activated to find out if it would be successful. This is important because it allows the schema mechanism to represent states that are not visible to the learning agent. In addition, since synthetic items can become part of the result for a schema, the schema mechanism can find ways to turn them *On* or *Off*.

The Schema Mechanism

Beginning the Learning Process

The schema mechanism begins with a set of *primitive items*, which are items that correspond to both coarse sensory input and proprioception that are updated automatically by the system. The primitive items are always either *On* or *Off* (only synthetic items may have the value *Unknown*). The schema mechanism is also endowed with a set of *primitive actions* that corresponds to simple movements, and each such primitive action also serves as the action for a schema with an empty context and an empty result, called a *bare schema*.

The sets of primitive items and primitive actions are user defined. Initially, everything that the schema mechanism knows about the world is expressed in the values of the primitive items and the only actions that the learning agent can take are the primitive actions. And, of course, the only schemas available to the learning agent are the bare schemas.

Learning begins by activating schemas to observe their results. Schemas compete for activation. The schema mechanism chooses schemas for activation while doing two fundamental activities: exploration and goal pursuit. There are two types of activation for schemas, *explicit activation* and *implicit activation*. Explicit activation is when a schema is selected for activation and its action is initiated. Implicit activation is when a schema's context happens to be satisfied and its action is initiated as part of activating some other schema that contains the same action. Both types of activation are used for learning.

The Microworld

Drescher tested his version of the schema mechanism using the microworld as shown in Figure 1 (image taken from (Chaput 2004)). This discussion of the microworld is presented here to give the reader a reference point that may prove helpful for grounding the information in the following sections.

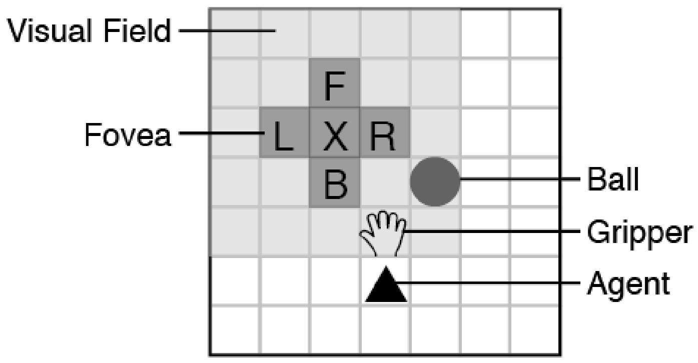

Figure 1: The microworld

In the microworld the agent cannot move but is endowed with ten primitive actions as given in Table 1. The hand can move in four directions in a 3×3 grid directly in front of the agent and occupy positions $(1,1)$ to $(3,3)$ (in Figure 1 the hand is at position (2,1)). The agent can also move its visual field in four directions within a 3×3 grid and occupy positions $(1,1)$ to $(3,3)$. A good way to think about this is that the agent can move its fovea, which is labeled "X" in Figure 1, in the same 3×3 grid as the hand (in Figure 1 the visual field is in position (1,3)). The learning agent can grasp something if it is to the left of its hand and can also ungrasp.

Table 1: Primitive Actions

handf	move hand front
handb	move hand back
handr	move hand right
handl	move hand left
eyef	move eye front
eyeb	move eye back
eyer	move eye right
eyel	move eye left
grasp	grasp object left of hand
ungrasp	open the hand

The primitive items endowed to the agent are given in Table 2. The agent can know the position of its hand and eye, and if its hand is closed and grasping something or just closed. The hand has four items for touch on its four sides, and four detail items for touch on its left side where its fingers are. So, for example if the ball were to the left of the hand, then tactl would be *On* and so would the detail item(s) for the ball. The body also has four items for touch, each on one side. Similar to the hand, if there is something directly in front of the body, then that item can be tasted. Note for both the fingers and the tongue that 2^4 different sensations are possible. The eye has 25 course visual items, so that if an object appears in that region then the corresponding item is *On*. And finally, each of the four foveal regions has 16 detail items for a total of 2^{16} different possible sensations for each foveal region. Using the configuration in Figure 1 as an

Table 2: Primitive Items

hp11, ..., hp33	hand positions
vp11, ..., vp33	visual positions
tactf	touching hand (front)
tactb	touching hand (back)
tactr	touching hand (right)
tactl	touching hand (left)
text0..., text3	detail touching fingers
bodyf	touching body (front)
bodyb	touching body (back)
bodyr	touching body (right)
bodyl	touching body (left)
taste0, ..., taste3	taste (in front of body)
hcl	hand closed
hgr	hand closed and grasping
vf00, ..., vf44	course visual field items
fovf00, ..., fovf33	front foveal region
fovb00, ..., fovb33	back foveal region
fovr00, ..., fovr33	right foveal region
fovl00, ..., fovl33	left foveal region
fovx00, ..., fovx33	center foveal region

```
SCHEMA-SELECTION
returns: schema s
  if activity = goal pursuit then
    select schema s randomly from those close to the maximum value, where schema value is based on a combination of primitive and delegated value in the schemas result, and added value for being an incomplete composite action in progress
  else
    select schema s randomly from those close to the maximum value, where value is based on a combination of hysteresis, habituation, action equalization, inverse action, and added value for being an incomplete composite action in progress
  end if
  if s is composite schema then
    return schema from composite action controller
  else
    return s
  end if
```

Figure 2: Pseudocode to Select a Schema for Activation

example, if taste2 and taste3 are triggered by the hand then the items hp21, vp13, vf41, bodyf, taste2 and taste3 are all *On*, and all other items are *Off*.

Selecting Schemas for Activation

The schema selected for activation determines what action the learning agent will take. The method used by the schema mechanism to select schemas for activation is rather complex; many factors are taken into account and Drescher does not explicitly give their relative importance. A confounding issue is that schemas with *composite actions* also select schemas for activation. A composite action is a high-level action with a *controller* that repeatedly selects actions until its goal is fulfilled. Composite actions and controllers will be explained in detail later.

As previously stated, the schema mechanism alternates between the broad activities of exploration and goal pursuit, both of which will be explained in more detail in the following two sub-sections. During both activities, a schema is selected for activation at each time step by the high-level selection mechanism. If a schema with a composite action is selected, then the composite action's controller selects the actual schema to be activated. The controller continues to select the schema to be activated at each time step as long as the composite action itself is still selected by the high-level mechanism at each time step. If that composite action controller in turn selects a schema with a composite action, then its controller selects the schema to be activated, and so on. Thus, there can be multiple levels of activation, but only one schema is actually activated at each time step.

Schemas are chosen based on activation importance within the exploration and goal pursuit activities. Within both activities a schema is chosen for activation randomly from those close to the maximum importance value, and to reduce thrashing, schemas are given added value if they are composite actions in progress. During exploration, the activation importance of schemas is based on what can be learned; and during goal pursuit, activation is based on reaching an explicit top-level goal. This explicit top-level goal is designated by an item with a value. This is separate from composite action goal attainment which will be discussed in in the section on composite actions. Further explanation is given in the following two sections and the high-level pseudocode can be found in Figure 2.

Goal Pursuit During goal pursuit the importance value of a schema is based on the *primitive*, *instrumental*, and *delegated* value of the items in its result. Primitive value is given to primitive items that are always useful to the learning agent. Examples include having an object centered in the fovea and having the hand touch an object. Instrumental value is given to items which are useful for achieving other things of value. Schemas that reliably chain to schemas with high value are given instrumental value. Instrumental value is only used during high-level goal pursuit, so it is not persistent because it depends on the current goal. Delegated value is like instrumental value except that it is persistent. To calculate delegated value, at each time step, the schema mechanism calculates the highest valued item that can be reached by a reliable chain of schemas starting with an applicable schema. For each item the schema mechanism keeps track of the average value of the highest value item reachable when the item is *On* and when it is *Off*. If this value is higher when the item is *On* compared with *Off* then the item gets positive delegated value, and it gets negative delegated value if the value is higher when *Off* then *On*. Delegated value did not play a large part in Drescher's implementation because the learning agent was at such a primitive level that there were no interesting things worth achieving. However, delegated value would take on importance in a more advanced implementation because it would allow the learning agent to assign value to synthetic items.

Exploration During exploration the schema mechanism seeks to learn about its world as opposed to trying to achieve certain goals, and mainly uses the concepts of hysteresis and habituation to choose schemas for activation. *Hysteresis* promotes repetition of a small number of tasks and provides a type of focus of attention. Schemas record their frequency of activation, and more frequently activated schemas are more likely to be selected for activation. This kind of "rich get richer" scheme allows a kind of specialization and could be used schema pruning (although pruning was not implemented by Drescher). *Habituation* allows the schema mechanism to move on to activating other schemas after a schema has been activated too many times. In addition, the schema mechanism also tries to spread out activation among actions. Finally, the schema mechanism identifies and promotes successive activation of inverse actions, meaning situations in which if one action turns an item *On* and then another action turns if *Off*. This allows the schema mechanism to find schemas that are locally consistent.

Marginal Attribution

New schemas are created by being *spun off* from existing schemas. To spin off a new schema means to create a copy of the existing schema and then to add the new item to its context (context spinoff) or result (result spinoff). Initially there are only bare schemas with no context and no result, but as learning progresses new schemas can be spun off from schemas that were themselves spun off.

Marginal attribution first finds results that may only be slightly more likely after an action (result spinoff) than otherwise, and spins off a new schema containing that result. This allows it to find results that may only occur in specific contexts without knowing what those context are. Marginal attribution then hillclimbs by spinning off new schemas with added context items eventually (it is hoped) culminating in a reliable schema (context spinoff). This hillclimbing works even in situations in which multiple context items are needed for a schema to be reliable because although the schema mechanism only examines adding one context item at a time, there will be many trials in which the other items will take their necessary values by chance, causing the increased reliability, even if slight, to be noticed. A disadvantage of this hillclimbing approach is that many intermediate, unreliable schemas are generated. Before a new schema is generated there is check that it does not already exist, but the schema mechanism has no garbage collection mechanism for unnecessary schemas, and such a mechanism would almost surely be needed in any realistic application.

Result Spinoff Schemas with new result items are added by result spinoff. Recall that each schema maintains an extended context and an extended result, both of which consist of statistics for each item. When a particular item in the extended result of a bare schema is even slightly more likely to be changed to either *On* or *Off* when a schema containing that action is activated than otherwise, a new schema containing that action and result is spun off from that bare schema. Note that since the result for a schema can only contain one item, only bare schemas participate in result spinoff. The statistics kept for each item in the extended result of each bare schema are the probability that the item turned *On* given that the action of the schema was taken divided be the probability that the item turned *On* given that the action was not taken, and the analogous statistic for *Off*. So for a bare schema s with action a and extended result item i the statistic for transition from *Off* to *On* is given by

$$\mathsf{s.prob}(\Delta^{+}\mathsf{i}|\mathsf{a})/\mathsf{s.prob}(\Delta^{+}\mathsf{i}|\neg\mathsf{a})$$

where

$$\mathsf{s.prob}(\Delta^{+}\mathsf{i}|\mathsf{a}) = \mathsf{s.count}(\Delta^{+}\mathsf{i}|\mathsf{a})/\mathsf{s.activated}$$

where the notation $\mathsf{s.count}(\Delta^{+}\mathsf{i}|\mathsf{a})$ is the number of times i turned *On* given that action a for s was taken, and $\mathsf{s.activated}$ is the number of times that action a for s was taken. The equation for $\mathsf{s.prob}(\Delta^{+}\mathsf{i}|\neg\mathsf{a})$ is analogous. Note that since bare schemas have no context items, a bare schema s is implicitly activated each time that its action is initiated by any other schema. The statistic for the transformation of item i to *Off* is analogous, and the pseudocode for updating both statistics is given in Figure 3.

When this statistic for an item i in schema s becomes greater than 1, then a new schema s' is created with the same

```
UPDATE-RESULT-STATISTICS
receives: schema $s$, item $i$, action taken $a$
• $s.count(\Delta^{+}i|a)$ is the number of times for schema $s$ that
  item $i$ turned On when action $a$ was taken ($\neg a$ is not
  taken) ($\Delta^{-}i$ indicates $i$ turned Off)

  if $s$ is not bare then
    return
  end if
  let $s_a$ = action from schema $s$
  if $s_a = a$ then
    if the value of $i$ turned On then
      add 1 to $s.count(\Delta^{+}i|a)$
    end if
    if the value of $i$ turned Off then
      add 1 to $s.count(\Delta^{-}i|a)$
    end if
  else
    if the value of $i$ turned On then
      add 1 to $s.count(\Delta^{+}i|\neg a)$
    end if
    if the value of $i$ turned Off then
      add 1 to $s.count(\Delta^{-}i|\neg a)$
    end if
  end if
```

Figure 3: Pseudocode for Updating Result Statistics

```
RESULT-SPINOFF
receives: schema s, item i
• s.count(Δ⁺i|a) is the number of times for schema s that
  item i turned On when action a was taken (¬a is not
  taken). (Δ⁻i indicates i turned Off.)
• s.activated is the number of times that the action for
  schema s was taken
• totalActivations is the total number of actions taken by
  the system

  if s is not bare then
    return
  end if
  let s.notActivated = totalActivations - s.activated
  let $prob(\Delta^+i|a) = s.count(\Delta^+i|a)/s.activated$
  let $prob(\Delta^+i|\neg a) = s.count(\Delta^+i|\neg a)/s.notActivated$
  let $prob(\Delta^-i|a) = s.count(\Delta^-i|a)/s.activated$
  let $prob(\Delta^-i|\neg a) = s.count(\Delta^-i|\neg a)/s.notActivated$
  if $prob(\Delta^+i|a)/prob(\Delta^+i|\neg a) > 1$ then
    copy schema s to s′ and add result item i=On
    CREATE-COMPOSITE-ACTION(s′) *** Fig. 9 ***
  end if
  if $prob(\Delta^-i|a)/prob(\Delta^-i|\neg a) > 1$ then
    copy schema s to s′ and add result item i=Off
    CREATE-COMPOSITE-ACTION(s′) *** Fig. 9 ***
  end if
```

Figure 4: Pseudocode for Result Spinoff

action as schema s and item i in the result of s′. The pseudocode for this is given in Figure 4.

Drescher actually weights both context and result statistics to more recent trials, but since this was an artifact of his implementation (although arguably useful) it will not be considered here.

To give an example, it may be that the door is more likely to go from closed to open when the OpenDoor action was taken compared to when the OpenDoor action was not taken, thus ⟨|OpenDoor|⟩ would spin off ⟨|OpenDoor|DoorOpen⟩.

One additional aspect is that the extended result of a schema is not updated for items that are changed due to being in the result of a reliable schema that was just activated. This keeps known causes from overshadowing other, possibly less robust, causes.

Context Spinoff Once the schema mechanism has created a schema with a result, it then looks for context items to make that result more reliable. When a particular item in the extended context of a schema is found to make a schema more reliable, then a new schema is spun off with that item added to its context. Two statistics are kept for each item in the extended context of each schema. The first is the ratio of the probability that the schema was successful given that the item was *On* to the probability that the schema was successful given that the item was *Off*. The second is the reciprical of the first. The first statistic for item i of schema s is given by

$$s.prob(success|i)/s.prob(success|\neg i)$$

where

$$s.prob(success|i) = s.count(success|i)/s.count(i)$$

and $s.count(success|i)$ indicates the number of times that s was successful when i=*On* and s.count(i) is the number of times that s was activated when i=*On*. The values for s where i=*Off* are analagous. The pseudocode for updating these values is given in Figure 5. If the first statistic in in the extended context of schema s goes above 1, then a new schema s′ with the added context item i = *On* is spun off from s. Again, the process for the success of the schema for when i is *Off* is analogous and the pseudocode for both is given in Figure 6.

Continuing the example, the schema mechanism may find that standing in front of the door makes ⟨|OpenDoor|DoorOpen⟩ more reliable and may spin off the new schema ⟨InFrontOfDoor|OpenDoor|DoorOpen⟩. The full pseudocode of the basic marginal attribution process is given in Figure 7.

Embellishments to Marginal Attribution In order to ensure that all correlations were found, and to tame the proliferation of schemas, Drescher had to add some additional constraints. One nice thing about the schema representation is that concepts that can be expressed with disjunctions can be represented with multiple schemas, each with the same action and same result but different contexts. However, when expressing disjunctive concepts as multiple schemas, the effect of some item values can be hidden. To counteract this and to help contain the proliferation of schemas,

```
UPDATE-CONTEXT-STATISTICS
receives: schema s, item i
• s.count(i) is the number of times that schema s was activated when item i was On (analogous for Off)
• s.count(success|i) is the number of times that schema s was successful when item i was On (analogous for Off)

  if s was activated (implicitly or explicitly) then
    if i is On then
      add 1 to s.count(i)
      if s was successful then
        add 1 to s.count(success|i)
      end if
    end if
    if i is Off then
      add 1 to s.count(¬i)
      if s was successful then
        add 1 to s.count(success|¬i)
      end if
    end if
  end if
```

Figure 5: Pseudocode for Updating Context Statistics

```
CONTEXT-SPINOFF
receives: schema s, item i
• count(i) is the number of times that schema s was activated when item i was On (analogous for Off)
• s.count(success|i) is the number of times that schema s was successful when item i was On (analogous for Off)
• $\theta_c$ is the threshold for context spinoff
  let prob(success|i) = s.count(success|i) / s.count(i)
  let prob(success|¬i) = s.count(success|¬i) / s.count(¬i)
  if prob(success|i) / prob(success|¬i) $> 1$ then
    copy schema s to s′ and add context item i=On to s′
  end if
  if prob(success|¬i) / prob(success|i) $> 1$ then
    copy schema s to s′ and add context item i=Off to s′
  end if
```

Figure 6: Pseudocode for Context Spinoff

```
MARGINAL-ATTRIBUTION
receives: action a
• s.activated is the number of times that the action corresponding to schema s was taken.

  *** Result Statistics and Spinoff ***
  for every bare schema s do
    if action for s = a then
      add 1 to s.activated
    end if
    for every item i do
      UPDATE-RESULT-STATISTICS(s,i,a)
      RESULT-SPINOFF(s,i)
    end for
  end for

  *** Context Statistics and Spinoff ***
  for every schema s do
    for every item i do
      UPDATE-CONTEXT-STATISTICS(s,i)
      CONTEXT-SPINOFF(s,i)
    end for
  end for
```

Figure 7: Pseudocode for Marginal Attribution

the marginal attribution mechanism defers to more specific schemas. It works as follows. Suppose the schema $\langle|a|r\rangle$ spins off a new schema with context item i creating $\langle i|a|r\rangle$. The embellishment then sets all of the the extended context statistics in $\langle|a|r\rangle$ to 0, and when item i is *On* $\langle|a|r\rangle$ does not update its statistics. This means that when i is *On* $\langle|a|r\rangle$ defers to $\langle i|a|r\rangle$, and so if there is another context item j then it will not be masked when i is *On*, and $\langle|a|r\rangle$ will recognize it and be able to spin off $\langle j|a|r\rangle$. Also, if context item k is generally *On* when i is *On*, and improves the reliability of result item r following action a, then it need only be spun off of $\langle i|a|r\rangle$ to create $\langle ik|a|r\rangle$. Without deferring to the more specific schema, $\langle|a|r\rangle$ would spin off $\langle k|a|r\rangle$. Thus, deference also helps to avoid the explosion of schemas.

A second embellishment of context spinoff is if multiple context items cross the threshold θ_c at the same time then the most specific one (the one that is *On* with the smallest frequency) is spun off.

Aggregating Context Items Results of schemas only contain one item. This is done to limit the explosive growth in the number of schemas. However, the chaining mechanism discussed in in the composite actions section can only find linear chains, meaning that results from multiple schemas cannot chain to one context. So, when a reliable schema has a context with multiple items, those items are aggregated to form one item in the extended result of all schemas. Thus, a schema can only have a result with multiple items if those items appear in the context of a reliable schema.

Override Conditions Sometimes, a schema needs an override to specify that it is not reliable. To use the example given by Drescher, assume that the reliable schema $\langle \mathsf{p}|\mathsf{a}|\mathsf{x}\rangle$ is not successful in the rare case when item w is *On*, and $\langle \neg\mathsf{wp}|\mathsf{a}|\mathsf{x}\rangle$ is spun off. Note that $\langle \mathsf{p}|\mathsf{a}|\mathsf{x}\rangle$ still exists and will not be successful when w is *On*, so an override on $\langle \mathsf{p}|\mathsf{a}|\mathsf{x}\rangle$ is given when w is on.

Synthetic Items

An important characteristic of the schema mechanism is that it is able to add to its ontology and to represent states that cannot be directly perceived. Recall that it does this by creating a *synthetic item* for any schema that is not *reliable* and is *locally consistent*. The schema that spawns the synthetic item is referred to as the *host schema* of the synthetic item. A synthetic item represents the circumstances that enable its host schema to be valid. For example, since doors that are able to be opened tend to stay that way for a while, and those that are locked tend to stay that way for a while, the unreliable and locally consistent schema

$$\langle \mathsf{InFrontOfDoor}|\mathsf{OpenDoor}|\mathsf{DoorOpen}\rangle$$

would spawn the synthetic item

$$[\mathsf{InFrontOfDoor}|\mathsf{OpenDoor}|\mathsf{DoorOpen}]$$

that could be called [DoorUnlocked]. (Note that none of these names mean anything to the schema mechanism, the synthetic item could just as well have been called SYMBOL14569.) Here, synthetic item names are enclosed in square brackets. This means that our running example has contained an abuse of notation; since its context and result items both refer to high-level states and would therefore be unlikely to be primitive in any implementation, our example is written properly as

$$\langle [\mathsf{InFrontOfDoor}]|\mathsf{OpenDoor}|[\mathsf{DoorOpen}]\rangle$$

This nicely shows how the schema mechanism can build increasingly higher levels of sophistication. The synthetic item representing the state of the world in which the door can be opened is now correctly represented as

$$[[\mathsf{InFrontOfDoor}]|\mathsf{OpenDoor}|[\mathsf{DoorOpen}]]$$

As with primitive items, a synthetic item can be added to the result of some schema, and therefore the learning agent could learn ways to make the synthetic item, and thus the expected validity of its host schema, turn *On* or *Off*. Additional features, such as seeing the latch in the crack of the door in our example, could also be found that determine whether or not the host schema is valid.

The values of the primitive items are given directly by the sensory apparatus of the system, and must be *On* or *Off*. The values of the synthetic items may be *Unknown* in addition to *On* or *Off*, and are given by the following four sources:

1. *Host schema trial.* When the host schema is activated, the synthetic item is given a value of *On* if it is successful and *Off* otherwise.

```
CREATE-SYNTHETIC-ITEM
• s.c_k(success) is the number of times schema s was successful during any window of k timesteps since a successful activation
• s.c_k(activation) is the number of times schema s was activated during any window of k timesteps since a successful activation
• s.count(success) is the total number of times schema s was successful
• s.count(activation) is the total number of times schema s was activated
• $\theta_{rel}$ is the reliability threshold
• $\theta_{lc}$ is the threshold for local consistency

  for every schema s do
     *** If Not Reliable ***
     if $s.count(success)/s.count(activation) < \theta_{rel}$ then
        *** If Locally Consistent ***
        if $s.c_k(success)/s.c_k(activation) > \theta_{lc}$ then
           make new synthetic item [s]
           add [s] to extended context and extended result of every schema
        end if
     end if
  end for
```

Figure 8: Pseudocode for Creating Synthetic Items

2. *Augmented context conditions.* A host schema may spin off a new schema that becomes reliable. A reliable schema must report its applicability to the schema that spun it off. If this reliable schema becomes applicable then the synthetic items for the parent schema is turned *On*.
3. *Predictions.* If a synthetic item appears in the result of a reliable schema, then if that schema is activated the synthetic item is turned *On* if positively included and *Off* if negatively included, unless there is evidence to the contrary.
4. *Local consistency.* When a synthetic item is turned *Off* or *On* it stays that way for a period of time specified by the duration of the host schema's local consistency before reverting to *Unknown*. This of course means that every unreliable schema must maintain the duration of its local consistency, or alternatively, a standard time period could be used.

The pseudocode for synthetic item creation is given in Figure 8.

Composite Actions

A *composite action* is like a subroutine, or an *option* (Sutton, Precup, & Singh 1999) in the reinforcement learning literature, it allows high-level actions to be performed. When a bare schema spins off a schema that has a result that is novel, meaning that no other schema has that result, then a composite action is created with that result as its goal state. The schema mechanism then creates a bare schema with that composite action as its action, which allows the schema mechanism to execute the action and to learn about the results of achieving the composite action's goal.

When a composite action is created, a *controller* for it is created. A controller has a slot for each schema to record the proximity of the schema to the goal state. When the controller is first initialized it does a search for all chains of reliable schemas that lead to the goal state and notes the distance between each reliable schema and the goal. This can be accomplished by a breadth first search (BFS) starting from the goal state. Initially, all schemas that have that item and value in their result are chained to the goal, and then the BFS continues on all schemas that chain to the schemas that chain to the goal. A schema s chains to schema s' if the item-value pairs of the context of s' are satisfied by the result item-value pairs of s. These reliable schemas that chain to the goal state are called the *components* of the composite action. A composite action is *enabled* when one of its components is applicable, and a composite action can only be selected for activation if it is enabled. When the composite action is activated the controller continuously activates the applicable schema closest to the goal until the goal is achieved or until the process times out.

Of course, when a composite action is first created, there will be no schemas that chain to the result other than the schema s_r, which when spun off initiated the creation of the composite action. Schema s_r will have no context and so no schemas can chain to it. Thus initially, a composite action will not be enabled. However, the controller initiates backchaining search periodically, and when reliable schemas are created that chain to the goal state they will be found and the composite action will become enabled.

To continue with the current example, when $\langle|\mathsf{OpenDoor}|\rangle$ spins off $\langle|\mathsf{OpenDoor}|[\mathsf{DoorOpen}]\rangle$, if $[\mathsf{DoorOpen}]$ is in the result of no other schema, then a composite action is created that has as its goal the state of the door being open. Eventually, as $\langle|\mathsf{OpenDoor}|[\mathsf{DoorOpen}]\rangle$ spins off $\langle[\mathsf{InFrontOfDoor}]|\mathsf{OpenDoor}|[\mathsf{DoorOpen}]\rangle$, or other reliable schemas are created with $[\mathsf{DoorOpen}]$ in their result, the controller will find these through backchaining. Thus, if the learning agent were not in front of the door but wished to open the door, then the controller would successively activate the closest applicable schema until the door became open or the composite action timed out.

Recall that a schema is implicitly activated if it is applicable and its action is taken due to the activation of another schema. Schemas with composite actions take this a step further, such a schema is implicitly activated if it is applicable and its goal state is achieved, even if that result had nothing to do with the actions of the learning agent. This, combined with the fact that the schema mechanism is more likely to activate schemas that were recently activated due to hysteresis, provide the schema mechanism with a behavioral tendency to imitate what happens in the environment.

Note that one minor constraint is that a schema with a composite action may not spin off a schema whose result in-

```
CREATE-COMPOSITE-ACTION
receives: newly created schema s

  if result r of s does not exist in any other schema then
    create composite action a_c
    create new bare schema with action a_c
    create composite action controller that contains the
    path distance from r for every schema (controller will
    periodically do BFS from r to find path distances)
  end if
```

Figure 9: Pseudocode for Creating Composite Actions

```
SCHEMA-MECHANISM
• totalActivations is the total number of activations

  loop
    obtain snapshot of item values
    s ← SCHEMA-SELECTION
    s_a ← action of s
    execute s_a
    add 1 to totalActivations
    MARGINAL-ATTRIBUTION(s_a)
    CREATE-SYNTHETIC-ITEM
  end loop
```

Figure 10: Pseudocode for the Schema Mechanism

cludes an item in the composite action's goal. This is done to reduce the number of redundant schemas created. The general pseudocode for composite actions is given in Figure 9, and the high-level pseudocode for the entire schema mechanism is given in Figure 10.

Achievements in the Microworld

Drescher applied the schema mechanism to the microworld shown in Figure 1. The first schema learned was simple grasping, $\langle|\mathsf{grasp}|\mathsf{hcl}\rangle$. It learned schemas that represented how shifting the learning agents glance moved objects in its visual field, e.g. $\langle\mathsf{vf21}|\mathsf{eyer}|\mathsf{vf11}\rangle$. It also learned schemas for moving both its hand and its eye, e.g. $\langle\mathsf{vp22}|\mathsf{eyeb}|\mathsf{vp21}\rangle$. Enough of each of these three sets of schemas were learned to form a network for each behavior, so for example, a chain of schemas could be found to move the hand from any position to any other position. Since a composite action is created each time a schema is created with a novel result, the learning agent could then move the hand or eye to any position from any other position by chaining schemas. It could also "move" an item in its visual field to any location in its visual field.

Additionally, schemas were learned for intermodal coordination. Schemas chained to $\mathsf{hp22}$ so that the learning agent could "suck its thumb" using the schema $\langle\mathsf{hp22}|\mathsf{handb}|\mathsf{taste1}\rangle$. The schema mechanism also learned what Drescher claimed were steps toward the concept of persistent objects with such synthetic items as [|hp23|tact1].

Evaluation of the Schema Mechanism

Strengths of the Schema Mechanism

The schema mechanism builds a model of how actions affect the world by creating reliable schemas. A reliable schema is created by finding a result that may be just slightly more likely after an action than otherwise, and then hillclimbing on context items to find situations in which the result reliably follows the action. Perhaps one of the most important aspects of the schema mechanism, however, is that it is able to add features to the state space allowing a more compact representation of the environment.

The schema mechanism learns an increasingly high-level representation by building alternating levels of synthetic items and composite actions. For example, since the synthetic item [DoorUnlocked] is treated like any other item, it can be placed in the result of some schema s by result spinoff allowing a composite action $\mathsf{a_c}$ to be created for it. The composite action $\mathsf{a_c}$ would have its own controller that would allow the learning agent to find chains of schemas that lead to the state of the world in which the door is unlocked. Additionally, a bare schema $\langle|\mathsf{a_c}|\rangle$ would be created, and if it were found to be unreliable and locally consistent, it itself would become a synthetic item. This creates a kind of scaffolding that the schema mechanism uses to continuously climbs from synthetic items to composite actions.

If there are n binary state variables, then there are 2^n different states, which is far too many to be represented by any type of DFA-based representation. An important observation is that for any particular action only a small number of the state variables are important in determining the outcome of that action. State representations used for DFAs, MDPs, and POMDPs have no obvious way of representing these "don't care" conditions. The schema method of Drescher avoids this difficulty by not representing whole states but instead only important substates.

Another important advantage, as pointed out by Drescher, that the schema method has over situation-action learning (e.g. reinforcement learning) is that the schema method learns continuously, as opposed to only when it happens to hit upon an action that elicits a reward. Additionally, he also points out that for situation-action learning that the current goal, or some indicator of the current goal, must be incorporated into the state, and that this increases the state space by a multiplicative factor.

Weaknesses of the Schema Mechanism

The most obvious and important weakness of the schema mechanism is computational complexity, this topic will be discussed in the section on computational complexity. Another shortcoming is the inability for the schema mechanism to represent any form of generalization such as $\langle \mathsf{hp(x)(y)|handb|hp(x)(y-1)} \rangle$. Drescher proposes a partial solution to this problem in the form of *virtual generalizations*, but it is not clear that this concept will eliminate this problem. Virtual generalizations will be discussed further in the section on possible future research directions.

Another disadvantage is that when chaining schemas the context of one schema can only be satisfied by activating a schema that has all of those item/value pairs in its result. The example given by Drescher is the goal of having two blocks on a table. The schema mechanism will not know to activate the schema for putting one block on the table twice, it will have to activate a schema that has in its result two blocks on the table. Thus if we had an item called (n)BOT meaning n blocks on the table, the final schema of the chain would have to look something like $\langle \mathsf{(n-1)BOT|PutBlock|(n)BOT} \rangle$. Essentially, the chaining mechanism requires that at each link of the chain the context of a schema in that chain must represent everything that has been achieved so far; and since there is no parameterization in the schema mechanism, (n)BOT cannot exist, and there would have to be a separate item for each number of blocks.

The Computational Complexity Problem We have seen that the schema mechanism has some interesting properties and a potential to be a useful learning architecture; however, the implementation proposed by Drescher is intractable. Since the schema mechanism can continuously create synthetic items and composite actions, the schema mechanism continues to do so until it runs out of memory, and at that point it can learn nothing further. In addition, the amount of time taken for each action grows with the number of schemas and items.

Drescher used a parallel architecture in his experiments, however since serial computers are much more common it is worth looking at the computational complexity of running the schema mechanism on a serial computer. With respect to time complexity, the biggest problems are marginal attribution, and the BFS used to find chains of schemas for instrumental value and the composite action controller. However, since the BFS is not necessary on every action the focus for time requirements will be on marginal attribution. The naïve implementation is given in Figure 7. If we let s be the number of schemas and i be the number of items then it takes $O(si)$ time.

However we can do it faster if we break it up into two loops as shown in figure Figure 11.

If we let i_δ be the number of items that have changed value, s_b be the number of bare schemas, and s_a be the number of schemas that contain action a then we get a running time that is $O(s_b i_\delta + s_a i)$, which is a great deal faster than $O(si)$. This is still a lot of work to do after each action, but it may be manageable.

A far bigger problem is space complexity. Drescher said that in his example run that the schema mechanism ran out of memory after creating just over seven thousand schemas. Each composite action has a controller that stores a value for each schema, but since most of the values in the composite action controller will be blank because only a small percentage of schemas chain to any result, the space used due to the composite action controller should be minimal. The real issue is that each schema stores statistics for each item in its extended context and extended result, thus the marginal attribution machinery takes space of $O(si)$.

```
FAST-MARGINAL-ATTRIBUTION
receives: action taken a

  *** Result Statistics and Spinoff ***
  for every bare schema s do
    if action for s = a then
      add 1 to s.activated
    end if
    for every item i that has just turned On or Off do
      UPDATE-RESULT-STATISTICS(s,i,a)
      RESULT-SPINOFF(s,i)
    end for
  end for

  *** Context Statistics and Spinoff ***
  for every schema s that uses action a do
    for every item i do
      UPDATE-CONTEXT-STATISTICS(s,i)
      CONTEXT-SPINOFF(s,i)
    end for
  end for
```

Figure 11: Pseudocode for Fast Marginal Attribution

Related Work on the Schema Mechanism

Replacing Marginal Attribution with SOMs

One method put forth to improve on the computational complexity of the schema mechanism is the Cognitive Learning Architecture Schema Mechanism (CLASM) (Chaput 2004). CLASM is a system of self-organizing maps (SOMs) (Kohonen 1995) is used to replace the marginal attribution mechanism of the schema mechanism.

In CLASM, a SOM is allocated for each primitive and composite action so that there is a one-to-one mapping between SOMs and actions. For each SOM, the weight vector of each node represents the extended context and extended result of the SOM's action and contains two times the number of items. To initialize the SOM, the extended context items are initialized to be between 0 and 1 and the extended result items are initialized to be between -1.0 and 1.0 (Chaput does not say how exactly to initialize those values, we can assume that it is done by some standard method).

For training, when an action is taken that action's SOM is trained on a vector consisting of the values of all the items before the action (extended context) and the change in the values after the action (extended result). For the extended context, items that are *On* are given a value of 1.0 and items that *Off* are given a value of 0.0. (He does not specify what to do with *Unknown* but we can assume that they are given a value of 0.5.) For the extended result, items that change to *On* after the action are given a value of 1.0, and those result items that change to *Off* are given a value of -1.0.

Training continues until all the action SOMs have stabilized. The SOMs are then harvested for schemas, which will have the action of the SOM. When a node of a SOM becomes a schema its weight vector is converted into a context and result for the schema. The context consists of all the vector context items whose value is greater than 0.9 (for *On*) and less than 0.1 (for *Off*), the other items are not used. The result consists of the vector result items whose value has changed by more than 0.9 and become *On* if the change was positive and *Off* if the change was negative, and again, the other items are not used. Only schemas with at least one result item are harvested.

Each harvested schema is then reified as a synthetic item and a composite action is created for each result item that appears in no other result as part of some schema. As in Drescher, duplicate schemas are not created. The resources of the current SOM level are then released and the training of the next level begins by creating a SOM for each action including the composite actions created in the previous level. Additionally, the extended context and extended result represented in each node then contains weights for all the previous items plus the new synthetic items. When CLASM was applied to the microworld of Drescher on a serial computer it was able to learn the same schemas as were learned in Drescher's implementation.

Computational Advantages of CLASM CLASM is a completely different way to do marginal attribution. It generates fewer unnecessary schemas because it eliminates the trail of intermediate hill climbing schemas. Thus, a schema with n context items can be added all at once as opposed to Drescher's implementation in which a chain of n schemas would have to be created. Additionally, CLASM eliminates many of the ad hoc methods that the schema mechanism uses to limit schema growth. For example, there is no need to keep track of whether one schema is more specific than another.

In CLASM, the time complexity after each action is better than in Drescher's implementation. Recall that in Drescher it was in $O(s_b i_\delta + s_a i)$ if i is the number of items, s is the number of schemas, s_b is the number of bare schemas, i_δ is the number of items that have changed value, and s_{a} be the number of schemas that contain action a. In CLASM since all that is necessary is to update a SOM, it is in $O(i)$.

With respect to space, the problem is a little more complex. Recall that the schema mechanism takes space of $O(si)$. CLASM creates a SOM for each action, and each node of each SOM contains values for each item, but since its schemas do not have to maintain an extended context or extended result, the space required is in $O(s + ai)$. However, in CLASM, since each harvested schema becomes a synthetic item, and many synthetic items are likely to end up in the extended result of some schema causing a composite action to be created, the number of actions may not be much less than the number of schemas.

Functional Differences Between CLASM and the Schema Mechanism Although CLASM appears to be in some ways more efficient, there are some important differences in behavior between CLASM and Drescher's implementation. In CLASM, if a particular item always has the same value, *On* for example, when an action is taken then the schemas created using that action will contain that item with value *On*, however in Drescher's implementation they

```
FM-MARGINAL-ATTRIBUTION-INITIAL
receives: action taken a

  *** Update Statistics ***
  for every schema s do
    if action for s = a and s is a bare schema then
      add 1 to s.activated
    end if
    for every item i that has changed value do
      UPDATE-RESULT-STATISTICS(s,i,a)
      UPDATE-CONTEXT-STATISTICS(s,i)
    end for
  end for

  *** Perform Spinoff ***
  for every schema s that uses action a do
    for every item i do
      RESULT-SPINOFF(s,i)
      CONTEXT-SPINOFF(s,i)
    end for
  end for
```

Figure 12: Pseudocode for the Foner and Maes Marginal Attribution Without Focus of Attention

will not because what is taken into account is the ratio of success of the schema when *On* compared with *Off*, and not just the correlation. Also, in CLASM it is not necessary for a group of items to be in the context of some schema in order to have that set consisting of more than one item in the result of some schema. In (Chaput 2004), value is propagated by using chaining, but it is not specified exactly how this is done in the absence of a constraint on chains being linear.

In CLASM, all schemas are converted to synthetic items once created as opposed to using Drescher's criteria of being unreliable and locally consistent. CLASM was only run for two levels, but in more complex environments this could cause the number of synthetic items to overwhelm the system. Additionally, as opposed to what was discussed in the section on synthetic items, synthetic items are turned *On* or *Off* only based on the result of activating the host schema. This may, however, have been due to expediency of implementation rather than on principle.

Reducing Computation with Focus of Attention

Another method put forth to reduce the computational complexity of the schema mechanism is described in (Foner & Maes 1994). It entails only performing the nested loops of marginal attribution on a limited number of schemas and items. They use the loop structure in Figure 12 as their initial baseline. They then reduce the amount of work in those loops by using two principles to focus attention. To reduce the work of updating the statistics, they use the principle of *perceptual selectivity*. For updating statistics they only loop on items that have had their value change in the last two clock ticks, and only on schemas that contain those items. They use the principle of *cognitive selectivity* to reduce the

```
FM-MARGINAL-ATTRIBUTION-FOA
receives: action taken a

  *** Update Statistics using Perceptual Selectivity ***
  let $i_{\delta 2}$ be items that have changed in last two clock ticks
  let $s_{\delta 2}$ be the schemas containing items in $i_{\delta 2}$ plus the
  bare schemas
  for every schema $s_{i\delta 2}$ do
    if action for s = a and s is a bare schema then
      add 1 to s.activated
    end if
    for every item $i_{\delta 2}$ do
      UPDATE-RESULT-STATISTICS($s_{\delta 2}$,$i_{\delta 2}$,a)
      UPDATE-CONTEXT-STATISTICS($s_{\delta 2}$,$i_{\delta 2}$)
    end for
  end for

  *** Perform Spinoff using Cognitive Selectivity ***
  let $i_\delta$ be items that have changed in last clock tick
  let $s_\delta$ be schemas with items in $i_\delta$
  for every schema $s_\delta$ do
    for every item $i_\delta$ do
      RESULT-SPINOFF($s_\delta$,$i_\delta$)
      CONTEXT-SPINOFF($s_\delta$,$i_\delta$)
    end for
  end for
```

Figure 13: Pseudocode for the Foner and Maes Marginal Attribution With Focus of Attention

work during spinoff. For those loops they only consider items which have changed in the last clock tick and only consider schemas whose item statistics have changed in the last clock tick. The pseudocode for marginal attribution using focus of attention is given in Figure 13.

They found that using focus of attention allowed the actions to be taken much faster than in the naïve implementation, but that it required about twice as many actions to learn the same schemas. However, given that the time saved increased as the robot learned more facts, they found that using focus of attention was important.

From Figure 13 we see that their running time for marginal attribution is $O(i_{\delta 2}s_{\delta 2} + i_\delta s_\delta)$ where $i_{\delta 2}$ is the number of items that have changed in last two clock ticks, $s_{\delta 2}$ is the number of schemas containing items that have changed in the last two clock ticks plus the bare schemas, i_δ is the number of items that have changed in last clock tick, and s_δ is the number of schemas with items that have changed in the last clock tick. Recall from the section on computational complexity that the running time for the marginal attribution as shown in Figure 11 is in $O(s_b i_\delta + s_a i)$ if i is the number of items, s is the number of schemas, s_b is the number of bare schemas, i_δ is the number of items that have changed value, and s_{a} is the number of schemas that contain action a. We see that the computational complexity of the pseudocode in Figure 11 could be reduced by using focus of attention on the items to reduce i from all items to $i_{\delta 2}$.

Schemas Compared with POMDPs and PSRs

A predictive state representation (PSR) (Littman, Sutton, & Singh 2001) is defined as a set of *tests* each consisting of a sequence of actions and observations that together provide sufficient information for a learning agent to know the results of all other possible tests. The state of the system is represented as a vector of probabilities of seeing the predicted observations given that the actions of the tests were performed. In (Littman, Sutton, & Singh 2001), it was shown that any environment that can be represented with a POMDP can also be represented with a PSR, and further that the number of the tests of the PSR would not be larger than the number of states of the POMDP.

In (Holmes & Isbell, Jr. 2005) a modified version of the schema mechanism was compared with PSRs. They modified the schema mechanism by replacing the statistic for result spinoff

$$\mathsf{prob}(\Delta^{+}\mathsf{i}|\mathsf{a})/\mathsf{prob}(\Delta^{+}\mathsf{i}|\neg\mathsf{a})$$

with

$$\mathsf{count}(\Delta^{+}\mathsf{i}|\mathsf{a})$$

where $\mathsf{count}(\Delta^{+}\mathsf{i}|\mathsf{a})$ is the number of times i turned *On* when action a was taken (both analogous for *Off*). They kept the statistic for context spinoff the same with the exception that they anneal the threshold. For the creation of synthetic items, they drop the requirement that the schema be locally consistent and only require that the schema be unreliable. Additionally, they only use host schema trials and predictions from reliable schemas to determine if a synthetic item is *On* or *Off*.

They found that their modified version of the schema mechanism worked better than the original schema mechanism on the benchmark POMDPs of flip, $\mathsf{float/reset}$, and $\mathsf{modified\ float/reset}$; $\mathsf{modified\ float/reset}$ is equivalent to $\mathsf{float/reset}$ except that the f action on the two right-most states leads deterministically to their left neighbor.

Additionally, they found that the modified schema mechanism was as accurate as PSRs on some of the POMDPs from (Singh *et al.* 2003). Of special note is that the modified schema mechanism took no more than 30,000 steps on any POMDP for sufficient learning, which was vastly fewer than the 1-10 million timesteps needed to learn the PSR parameters (Singh *et al.* 2003).

These results appear to show that the schema mechanism (at least in modified form) is equivalent in representational power to the established method of POMDPs. However, the schema mechanism was not designed for such artificial environments with such a high amount of state ambiguity. Therefore, the schema mechanism's next important milestone may come from an implementation that allows it to learn something useful in a more realistic environment. Such possibilities for future work will be discussed in the next section.

Possible Future Research Directions

Improve on CLASM

CLASM appears does a good job of limiting the amount of work that must be done after each action, especially if some measures are taken to constrain the growth of synthetic items. However, CLASM still uses memory proportional to the number of actions times the number of items. One way to reduce the memory under the CLASM framework would be to limit the number of action SOMs using memory at any one time. One such scheme would be to only create an action SOM the action corresponding to that SOM was executed. The SOM could then stay in memory for some fixed period of time and its resources could be released if the action is not executed again in that time period.

Replace or Enhance the Schema Activation Mechanism

The method for determining which schemas to activate is rather complex. Since the publication of Drescher's book this issue of intrinsic motivation has received a great deal of attention. For example, in (Oudeyer *et al.* 2005) through the method of Intelligent Adaptive Curiosity (IAC), the robot constantly seeks to explore in the area where it can learn the most. It avoids areas that are far too difficult and areas with which it is already familiar to find the part of its environment that enables the most learning. Something similar could be employed in the schema mechanism by using the reliability of the schema to affect its chances for activation. Schemas that are moderately reliable could be favored over very reliable or very unreliable schemas.

Virtual Generalizations

The schema mechanism has no way of representing generalizations such as $\langle \mathsf{hp(x)(y)}|\mathsf{handb}|\mathsf{hp(x)(y-1)}\rangle$. However, Drescher proposes using canonical frames of reference which he refers to as *virtual generalizations*. This is similar to how deictic references are used in (Ballard *et al.* 1996), and also is similar to finding islands in state space search in classical artificial intelligence.

For example, to move an object forward with the hand the learning agent can center the object in the fovea by moving its glance and then always use always the schema $\langle \mathsf{hgr\&vf22\&SeeHand@32}|\mathsf{handf}|\mathsf{vf23}\rangle$ where $\mathsf{SeeHand@32}$ is the conjunction of items associated with seeing the hand at that glance-relative position. This use of virtual generalizations is not an explicit capability that must be built into the system, but rather it can emerge in a sufficiently sophisticated implementation due to the instrumental value of having an object in the fovea.

Conclusion

The schema mechanism has shown potential to be a viable robot learning system. Given that proof of concept implementations that have been achieved in very simple environments, a logical next step is to implement the schema mechanism in a more complex environment where it can learn to do interesting things. Although much work has been done on alleviating the burden of computational complexity, more progress is needed. Additionally, an implementation in a larger environment would almost certainly require that more tradeoffs, such as focus of attention and keeping a limited number of actions in memory, be explored. However, considering recent work by (Chaput 2004) and (Holmes & Isbell, Jr. 2005), momentum seems to be building, and such an implementation may be undertaken in the not so distant future.

References

Ballard, D. H.; Hayhoe, M. M.; Pook, P. K.; and Rao, R. P. 1996. Deictic codes for the embodiment of cognition. *Behavioural and Brain Science*.

Chaput, H. 2004. *The Constructivist Learning Architecture: A Model of Cognitive Development for Robust Autonomous Robots*. Ph.D. Dissertation, University of Texas at Austin, Department of Computer Sciences. Also available as UT AI TR04-34.

Drescher, G. L. 1991. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA: MIT Press.

Foner, L., and Maes, P. 1994. Paying attention to what's important: Using focus of attention to improve unsupervised learning. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94)*.

Holmes, M. P., and Isbell, Jr., C. L. 2005. Schema learning: Experience-based construction of predictive action models. In Saul, L. K.; Weiss, Y.; and Bottou, L., eds., *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press. 585–592.

Kohonen, T. 1995. *Self-Organizing Maps*. Berlin; New York: Springer.

Littman, M. L.; Sutton, R. S.; and Singh, S. P. 2001. Predictive representations of state. In *NIPS*, 1555–1561.

Oudeyer, P.-Y.; Kapalan, F.; Hafner, V.; and Whyte, A. 2005. The playground environment: Task-independent development of a curious robot. In *AAAI Symposium on Developmental Robotics*.

Piaget, J. 1952. *The Origins of Intelligence in Children*. New York: Norton.

Piaget, J. 1954. *The Construction of Reality in the Child*. New York: Ballantine.

Singh, S.; Littman, M. L.; Jong, N. K.; Pardoe, D.; and Stone, P. 2003. Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning*.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112(1-2):181–211.