An Architecture for Adaptive Algorithmic Hybrids

Nicholas Cassimatis, Perrin Bignoli, Magdalena Bugjaska, Scott Dugas, Unmesh Kurup, Arthi Murugesan, and Paul Bello

***Abstract*—We describe a cognitive architecture for creating more robust intelligent systems. Our approach is to enable hybrids of algorithms based on different computational formalisms to be executed. The architecture is motivated by some features of human cognitive architecture and the following beliefs: 1) Most existing computational methods often exhibit some of the characteristics desired of intelligent systems at the cost of other desired characteristics and 2) a system exhibiting robust intelligence can be designed by implementing hybrids of these computational methods. The main obstacle to this approach is that the various relevant computational methods are based on data structures and algorithms that are difficult to integrate into one system. We describe a new method of executing hybrids of algorithms using the *focus of attention* of multiple modules. The key to this approach is the following two principles: 1) Algorithms based on very different computational frameworks (e.g., logical reasoning, probabilistic inference, and case-based reasoning) can be implemented using the same set of five *common functions* and 2) each of these common functions can be executed using multiple data structures and algorithms. This approach has been embodied in the Polyscheme cognitive architecture. Systems based on Polyscheme in planning, spatial reasoning, robotics, and information retrieval illustrate that this approach to hybridizing algorithms enables qualitative and measurable quantitative advances in the abilities of intelligent systems.**

***Index Terms*—Hybrid architectures, integrated systems.**

I. Introduction

We DESCRIBE a cognitive architecture for creating more robust intelligent systems by executing hybrids of algorithms based on different computational formalisms. There are several properties that we desire of intelligent systems. Each of these properties is exhibited by some algorithm, but often at the cost of one of the other properties. For example, many search algorithms and many probabilistic inference algorithms are general insofar as a wide variety of problems can be reformulated so that these algorithms can solve them. They are flexible in that, when small changes are made to the knowledge bases or models these methods use, they correctly update their inferences. However, for larger problems, these algorithms become prohibitive in time and/or space. They thus trade efficiency for scalability. Reactive and behavior-based systems trade generality for speed and dynamism. These systems can quickly adapt their behavior to new information about the environment but are generally not capable of making many kinds of complex inferences and plans that many reasoning or planning algorithms can. Approaches that are based on more complex and structured knowledge representation schemes, such as frames [1], scripts [2], and cases [3], make the same tradeoff. They can make inferences and plans that would take more general algorithms too much time. However, frames, scripts, and cases often do not work in situations that are even slightly different from those for which they were created.

Because of such tradeoffs, it has been difficult to create a single system that exhibits all the desired characteristics of intelligent systems. One approach to this problem is to create systems based on hybrids of these algorithms. This paper presents a cognitive architecture, called Polyscheme, for implementing and executing such hybrids. Polyscheme's design is motivated by some aspects of human cognitive architecture and several computational considerations. Many methods of integration involve loosely coupled collections of modules or hybrids of only a small fixed set of algorithms. Although Polyscheme includes modules that can encapsulate algorithms, it also enables algorithms to be implemented through sequences of "attention fixations" (each involving all the modules) called "focus traces." By interleaving these focus traces, Polyscheme can execute hybrids of algorithms where each step of each encapsulated algorithm can potentially be assisted by the others. As explained hereinafter, this form of integration enables systems that have previously not been feasible.

II. Unifying Principles

We can motivate an architecture for integrating hybrids of multiple classes of algorithms by recognizing that, even though they are based on very different computational formalisms, they share many common elements.

The following are some formal preliminaries. Strings of the form $P(\ldots x_i \ldots, t, w)$ are called propositions. They say that relation P holds over arguments x_i during temporal interval t in world w. Any kind of entity can be an argument. A world is a history (past, present, and future) of states. P/w refers to a proposition like P except for having w as its world argument. E ("eternity") is the temporal interval such that all other temporal intervals occur during it. R is the real world. Terms can refer to the same object. This is indicated with $Same(x, y, E, w)$. T, F, and U ("unknown") are the truth values for propositions.

Propositions are used only to characterize certain aspects of Polyscheme and as an interlingua between Polyscheme modules. Polyscheme is not a "logical" system insofar, as it can use nonlogical data structures and its computation is not predominantly deductive or confined to manipulating formulas in some logical language.

Manuscript received August 13, 2008; revised February 10, 2009 and June 5, 2009. This paper was recommended by Associate Editor M. Huber.

N. Cassimatis, P. Bignoli, M. Bugjaska, S. Dugas, U. Kurup, and A. Murugesan are with Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: cassin@rpi.edu; bigniop@rpi.edu; bugjam@rpi.edu; dugass@rpi.edu;
AQ1 kurupu@rpi.edu).

P. Bello is with the Office of Naval Research, Arlington, VA 22203-1995
AQ2 USA (e-mail: paul.bello@navy.mil).

Digital Object Identifier 10.1109/TSMCB.2009.2033262

A. Common Functions

It is possible to characterize algorithms from very different formal frameworks as executing sequences of common functions. In this case, "function" refers not to a mathematical function but to the purpose of an algorithm as in "the function of a sorting algorithm is to order the elements of a collection." These functions are described along with the notation we will use to refer to them.

1) Store Information: Given the information about the truth value of P, store it. $\mathtt{Store}(\mathtt{P}, \mathtt{TV})$.

2) Offer Opinion: Return a truth value for a proposition. $\mathtt{OpinionOn}(\mathtt{P}, \mathtt{TV})$.

3) Forward Inference: Given some knowledge, infer what follows. $\mathtt{ForwardInference}(\mathtt{BK})$, where $\mathtt{BK} = \{\ldots(\mathtt{P}_i, \mathtt{TV}_i)\ldots\}$ returns a list of propositions paired with their respective truth values.

4) Request Information/Subgoal: In order to know about something, take actions to learn about it. Given P, return a set of propositions such that information about their truths will help infer the truth of P. $\mathtt{RequestInformation}(\mathtt{P})$.

5) Identity: For any object, find other objects to which it might be identical. Where O refers to an object, BK is as mentioned before, and W is an alternate world, return a set of objects that might be identical to O in W. $\mathtt{Matches}(\mathtt{O}, \mathtt{BK}, \mathtt{W})$

Algorithm 1. Gibbs Sampling
Represent a state variable as a proposition. Let $\{\ldots\mathtt{P}_i\ldots\}$ be the state variables.
Start with an assignment of truth values to those propositions $\{\ldots(\mathtt{P}_j/\mathtt{w}_0, \mathtt{TV}_i)\ldots\}$.
for $i = 1$ to MAX-SIMULATIONS:
  for each j:
    $\mathtt{Store}(\mathtt{Pj}, \mathtt{ForwardInference}(\{(\mathtt{P1}/\mathtt{wj}-1,$
    $\mathtt{OpinionOn}(\mathtt{P1}/\mathtt{wi}-1))\ldots(\mathtt{Pj}-1/\mathtt{wi}-1,$
    $\mathtt{OpinionOn}(\mathtt{Pj}-1/\mathtt{wi}-1)))$
$\mathtt{Prob}(\mathtt{P}) = \textit{the proportion of worlds}$ w, in which P/w is true.

6) Represent Alternate Worlds: The ability to represent and make inferences about alternate states of the world is implicit in all of the aforementioned common functions since they all involve propositions with worlds as arguments.

The following are examples of how some important algorithms can be implemented using common functions.

Algorithm 2. WalkSAT.
for each proposition, P, $\mathtt{Store}(\mathtt{P}, \mathtt{Random}(\mathtt{T}, \mathtt{F}))$
for $i = 1$ to MAX-FLIPS
if $\mathtt{OpinionOn}(\mathtt{C}) = \mathtt{T}$
  Return $\{\ldots(\mathtt{Pi}, \mathtt{OpinionOn}(\mathtt{Pi})\ldots\}$
else
  with probability p,
    $\mathtt{Store}(\mathtt{P}, \mathtt{not}(\mathtt{OpinionOn}(\mathtt{R})))$
otherwise
  $\mathtt{Store}(\mathtt{ForwardInference}(\mathtt{BK}).\mathtt{first}())$,
    where BK is the set of ordered pairs of each proposition with its truth value.

A variant of Gibbs sampling [4] can be implemented as follows, where $\mathtt{ForwardInference}(\mathtt{BK})$ samples a value for P given the value of variables in its Markov blanket in BK.

Stochastic local search performed, for example, by the WalkSAT algorithm [5], can be used to find solutions to Boolean constraint satisfaction problems. Many problems (e.g., planning, diagnosis, and circuit design) can be mapped to satisfiability problems (or weighted variants of them). When the following focus scheme is in place, Polyscheme performs WalkSAT for a constraint C if $\mathtt{ForwardInference}(\mathtt{BK})$ returns a single ordered pair $(\mathtt{P}, \mathtt{TV})$, where P is the proposition such that flipping its truth value will make the most clauses in C true.

Algorithm 3. DPLL.
$DPLL(P, t\nu)$
$w \leftarrow$ world from the world P by assuming that the truth value of P is $t\nu$.
if $\mathtt{ForwardInference}(\mathtt{P}/\mathtt{w})$ returns a contradiction
  return false.
if all constraints in w are satisfied
  return true.
$V \leftarrow \mathtt{RequestInformation}(\mathtt{P}/\mathtt{w})$.
return $DPLL(V, true) \square DPLL(V, false)$.

The modern variants [6], [7] of the Davis–Putnam–Logemann–Loveland (DPLL) [8] algorithm are among the fastest complete satisfiability algorithms known. DPLL performs depth-first search through the space of assignments of truth values to variables. As each assignment is made, DPLL performs an elaboration step that makes assignments that follow from the existing assignments or can be assumed without contradiction. This elaboration step eliminates many impossible assignments from being explored and, thus, in many cases, significantly speeds search. If we assume that ForwardInference performs the DPLL elaboration step, then DPLL can be reformulated as in Algorithm 3. Finally, case-based reasoning (CBR) can be added to WalkSAT by modifying ForwardInference mentioned earlier to return propositions that were true in situations whose similarity to the current situation exceeds some threshold.

These examples illustrate how algorithms developed within very different formal frameworks can be executed with sequences of calls to the same five common functions.

The fact that multiple algorithms from different subfields based on different formal frameworks can all be executed as sequences of common functions motivates an approach to integrating them. We can create hybrids of two algorithms by interleaving the sequences of common functions that execute each algorithm.

When an algorithm performs a common function on a proposition, we say that it *attends to* or *fixes its attention* on this proposition. We call this event an *attention fixation.* Each of the algorithms described earlier attends to (i.e., executes a common function on) one proposition at a time.

The sequence of attention fixations that an algorithm makes when it executes is called its *focus trace*. Focus traces thus provide a uniform way of characterizing the execution of algorithms from different computational methods.

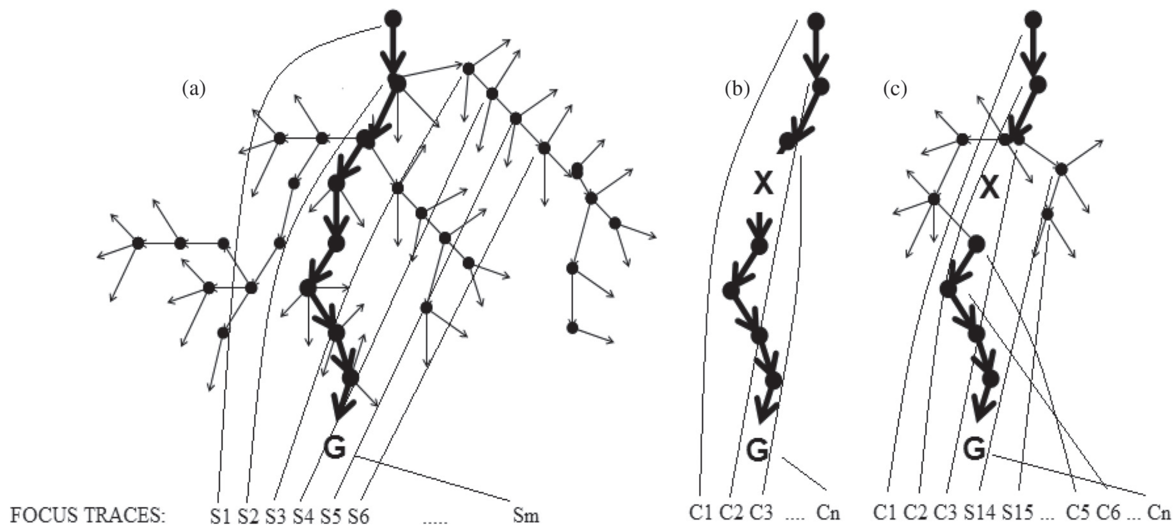

Fig. 1. Algorithms and their focus traces. The thick arrows depict a successful path to a goal found by a hypothetical run of a search algorithm. The thin arrows depict other fixations explored during the search for the best path. The (C) focus trace for the hybrid of CBR and search is a combination of the focus traces for (A) search and (B) CBR. These combine to surmount the problem (depicted by "X") by simply applying the case.

Interleaving the focus traces from two algorithms amounts to executing a hybrid of those algorithms. Specifically, an algorithm H is a hybrid of algorithms A1 and A2 if the focus trace of H includes fixations from A1 and A2. Fig. 1 shows the hybrid execution of CBR and search. The focus trace (1c) for the hybrid of CBR and search is a combination of the focus traces for search (1a) and CBR (1b).

B. *Multiple Implementation*

In each of the examples used to illustrate the common function principle, the common functions were implemented differently. The ability of a surprisingly diverse collection of computational methods to implement each of the common functions is what we call the *multiple implementation principle*. It is a key to enabling the integration of data structures and algorithms described herein. The following examples lend support to the multiple implementation principle.

1) Storing Information: Both rule-based systems and neural networks store information. Rule matchers must keep track of which propositions they have been given as input and which they have asserted as the result of rule matches. Neural networks that implement content addressable memories store patterns using the weights of the connections between units in the network. For example, the eigenvectors of the matrix representing unit connections in Hopfield [9] neural networks, for example, correspond to patterns stored by those networks. Thus, the function of storing information is a very simple example of how very different algorithms and data structures can implement the same common function.

2) Forward Inference: Both rule-based systems and feedforward neural networks, although based on very different data structures and algorithms, take inputs and produce outputs. Forward chaining in rule-based system matches the left-hand side of a rule against a set of propositions and asserts new propositions. Likewise, many forms of neural networks can be characterized as performing inference [10]. The values of input and output layers of neural networks can be represented as propositions [e.g., a unit representing that the temperature is 25 °C can be represented using the proposition *Temperature(25)*]. Thus, just like rule-based systems, feedforward neural networks can be characterized as taking propositions (those representing the values of the input units) and producing new propositions (those representing the values of the output units).

3) Subgoaling: Backward chaining in rule-based systems, subgoaling in logic-theorem provers, and subgoaling in means-end planners are obvious instances of the subgoaling common function. Less obviously, to learn the truth values of propositions representing the output units of neural networks, one can perform a kind of subgoaling by finding (the truth values of propositions representing) the values of the input units. Even directing visual attention is a form of subgoaling. In this case, to learn about the truth value of a proposition representing the attribute of an object, a goal is made of directing a camera or a pair of eyes toward that object.

4) Identity Matching: Algorithms used in object recognition (e.g., Bayesian networks, neural networks, and support-vector machines) all are performing a kind of similarity matching by taking a visually perceived object and finding the most similar stored object. Case-indexing and retrieval schemes in CBR have the same function.

5) Representing Alternate Worlds: Mechanisms which can make inferences about the real world can make inferences about hypothetical worlds.

These examples illustrate that common functions can be implemented using computational methods from very different subfields of artificial intelligence.

Implementing a single common function with multiple different algorithms and data structures enables another kind of hybridization. Consider an algorithm `A` executed in a particular way using common functions `C1...Cn`. If each of the `C` is executed using multiple computational methods `M1...Mn`, then every step of the execution of algorithm `A` will also involve the methods `M`.

As another example, one could imagine Gibbs sampling executed using a sequence of common functions, each of which is executed by a neural network. Thus, although the neural network would be performing all the computations (in this case, computing the likelihood of a proposition given its Markov blanket), the focus of Polyscheme can be selected in such a way that it implements Gibbs sampling. If one of the `Mi` executing a common function is an algorithm processing new sensor information, then every step of inference executing using this common function would incorporate up-to-date information from the world.

C. *Cognitive Self-Regulation*

The common function principle shows that very different kinds of algorithms can be executed as sequences of the same set of common functions. How does an intelligent system implementing more than one algorithm decide which common functions to choose? A recurrent theme among the algorithms used to illustrate the common function principle is that they choose which common function to execute in response to *metacognitive problems*. This is evident in several broad classes of algorithms.

Search algorithms choose possible worlds by assuming the truth of propositions that are *unknown*. If the truth value of a proposition is known, it is considered fixed, and worlds with that proposition having the opposite value are not explored. In the case of constraint-based search algorithms, the *metacognitive* problem is *ignorance*, which occurs when the truth value of a proposition is not known. In the case of search-based planning, the ignorance is often based on *conflict* since there is more than one possible next action to take or explore.

Many stochastic simulation algorithms for probabilistic reasoning (e.g., Gibbs sampling) also choose worlds to explore based on unknown values of a state variable. However, the level of ignorance in this case is somewhat reduced since a probability distribution for the state variable is known. This leads to a somewhat different strategy of exploring worlds: Worlds with more likely propositions being true are likely to be explored more often.

CBR algorithms are also often driven by ignorance and differ from search and stochastic simulation algorithms by how they react to ignorance. Instead of exploring possible worlds where different actions are taken, they retrieve similar solutions to similar problems and try to adapt them.

Thus, CBR, search, and stochastic simulation are different ways of addressing different kinds of *metacognitive problems*. We call the insight that many algorithms can be characterized by which common functions they choose in response to metacognitive problems the *cognitive self-regulation principle*.

This analysis resembles Soar's [11] implementation of universal weak methods by reacting to impasses (analogous to metacognitive problems) by reasoning in problem spaces (analogous to alternate worlds). However, together with the common function and multiple implementation principles, the cognitive self-regulation principle motivates a significantly different architecture.

III. POLYSCHEME

The principles[1] from the last section motivate the design of a cognitive architecture called Polyscheme. In Polyscheme, algorithms can be implemented as strategies for focusing the attention of multiple modules. By combining strategies, we lay down hybrid focus traces (as described in the last section) and thus execute hybrids of algorithms implemented as focus control strategies. By using modules based on different data structures and algorithms, we enable hybrids of algorithms implemented as focus control strategies and the algorithms in the modules. By including modules that can sense the world and by tightly bounding the time each focus of attention takes, inference can constantly use and react to the latest information from the world. This section and the next elaborate these points.

A Polyscheme system is comprised of a set of specialist `S`, an attention buffer `A`, and a focus manager `FM`. `FM` implements a `getNextFocus()` procedure that returns the next proposition to attend to. All the specialists must implement the following procedures: `Store()`, `OpinionOn()`, `Matches()`, and `ModifyFM()`. The latter influences which propositions `FM` selects. It will be described in more details later in this section.

At every time step, `FM` chooses a proposition for all the specialists to "focus on." Specifically, specialists give their opinions on the proposition's truth value, get all the other specialist's opinions on it, make their own inferences based on this new information, and request other propositions for the `FM` to focus on. The reasons why all specialists focus on the same proposition at the same time are as follows: 1) so that no specialist makes inferences based on a truth value that another specialist knows to be incorrect and 2) because, even though a proposition is focused on because specialist `S1` requested to focus on it, it might become relevant for some other specialist `S2`.

More formally, the Polyscheme control loop is

```
Do forever :
  "Select the next focus."
  P = FM.getNextFocus().
  "Get each specialists opinion on the focal prop."
  For all specialists S :
    TVi = S.opinionOn(P)
  "Inform specialists of each other's opinions."
  For all specialists S :
    For each of the TVi
      S.Store(P, TVi)
  "Allow specialists to influence the Focus Manager."
    For the focus queue Q and all specialists S :
      S.modifyFM(P, Q)
```

Once a focus of attention is chosen, all the specialists must focus on and therefore execute the same functions on it.

[1]There are also several aspects of human psychology that motivate the main aspects of Polyscheme. These are described in [12] and [13].

Therefore, the choice of attention fixation of the `FM` controls the flow of computation. We have experimented with several kinds of `FM`s but will use a very simple `FM` based on a queue (`FQ`) to illustrate how attention selection can implement hybrids of many kinds of algorithms. Although achieving all our long-term objectives will almost certainly require a more sophisticated focus manager, a queue-based focus manager has been sufficient to achieve significant results and demonstrate the promise of the approach. Thus, in what follows, `ModifyFM`() takes a queue as input and modifies it. These modifications can involve adding elements, deleting them, and changing their order.

The principal means of interaction among specialists is the sharing of opinions in Polyscheme's control loop. All specialists are first asked their opinion on the focal proposition, and then, each learns about these opinions and processes them. Therefore, specialists must wait for all of other specialists to finish offering their opinion before they proceed to use these opinions. This keeps specialists from making an inference on the opinion of one specialist that is contradicted by another specialist during the same time step.[2] If a proposition's truth value is inferred by a specialist during one iteration of the loop, other specialists will only learn about it if that proposition is focused on at a subsequent time step.

To illustrate, consider a Polyscheme system that has a rule specialist with $A \rightarrow B$, a perception specialist that can see that A and C are true, and a neural network specialist that classifies B and C as a situation represented by the proposition D. This licenses the following inferences: B is true (because A is true), and thus, D is true (because B and C are classified as a situation where D is true). The following is a sketch of how information would flow through the focus of attention in Polyscheme to generate these inferences.

1) The perception specialists puts A and C on the focus queue.
   a) Summary: The perception specialist requests that specialists focus on what it has seen to be true.
2) At a later iteration of the control loop, A is chosen for focus.
   a) Summary: A is focused on, and perception specialist says it is true; rule specialist infers that B is true and requests for focus on B.
   b) Taking opinions
      i.The perception specialist asserts its opinion that A is true.
   c) Storing opinions
      i.The rule specialist infers that B is true.
   d) Requesting focus
      i.The rule specialist puts B on the focus queue.
3) At a later iteration, B is chosen for focus.
   a) Summary: "B is focused on, rule specialist asserts it is true (because of the rule $A \rightarrow B$), and the neural network specialist commits this to memory."
   b) Taking opinions
      i.The rule specialist asserts its opinion that B is true.
   c) Storing opinion
      i.The neural network specialist stores in its memory that B is true.
   d) Requesting focus
      i.None of the specialists infers anything new; thus, the focus queue is not changed.
4) At a later iteration of the control loop, C is chosen for focus.
   a) Summary: "C is focused on, and perception specialist asserts it is true; neural network classifies this as situation D and requests that D be focused on."
   b) Taking opinions
      i.The perception specialist asserts its opinion that C is true.
   c) Storing opinions
      i.The neural network specialist stores in its memory that C is true and classifies this as a situation D.
   d) Requesting focus
      i.The neural network specialist requests focus on D.
5) At a later iteration, D is chosen for focus.
   a) Summary: "D is focused on, the neural network specialist asserts it is true, and all the other specialists learn this."
   b) Taking opinions
      i.The neural network specialist asserts its opinion that D is true.
   c) Storing opinions
      i.All the specialists learn that D is true.

This example illustrates that, if a specialist makes an inference about proposition P at one time step, it is only learned by other specialists after P is focused on at a future time step. This is a relatively cumbersome process for making what intuitively appears to be a simple two-step inference. However, as we will demonstrate hereinafter, implementing inference through the focus of attention of multiple specialists can thus generate significant benefits.

IV. Algorithms as Focus Control Strategies

In this section, we show how to implement algorithms using focus management in Polyscheme and illustrate the integration that this enables. We will concentrate mostly on algorithms from Section II. In that section, algorithms were described using common functions. In this section, specialists implement those common functions. We illustrate how focus control chooses which propositions those common functions operate on. In each case, a key component of the implementation is how the `ModifyFM` procedure is implemented. In the description of this procedure for each algorithm, we use italics to emphasize the role of a metacognitive problem (such as uncertainty, conflict, or ignorance) in choosing the focus of attention. This illustrates how cognitive self-regulation (through the choice of

[2]This implies that, if a specialist finishes computing its opinion before other specialists, it must waste time waiting for them to finish. In practice, two factors mitigate this waste. First, when a system is run on one or a few processors, idle CPU time is allocated to specialists not done computing, and there is little waste. Second, specialists are often designed to return their opinions very quickly. This minimizes the time they spend waiting for other specialist to finish. Such waste as there is, however, is compensated for significantly by the fact that specialists will never make inferences based on information another specialist knows to be false. The results reported later in this paper demonstrate that, despite this waste, we can still achieve significant performance improvements.

attention fixation) is a common aspect of the flow of control in the algorithms we describe.

First, let us consider how to implement a local-search algorithm such as WalkSAT. We can implement pure WalkSAT using a "constraint specialist." At any given time, this specialist encodes a constraint C in a conjunctive normal form. The specialist's functions operate as follows:

$$\mathtt{Store}(\mathtt{P},\mathtt{TV}): \mathtt{P} \text{ and } \mathtt{TV} \text{ are added as a clause to } \mathtt{C}.$$

`ModifyFM(P, Q)`: If `C` is satisfied in the world w of `P`, then add `Satisfied(C, E, w)` to Q. Otherwise, put the proposition `P` involved in most clauses with *conflicting truth values* at the front of the focus `Q`

$$\begin{aligned}&\mathtt{OpinionOn}(\mathtt{P}): \text{If } \mathtt{P} \text{ is of the form}\\ &\mathtt{Satisfied}(\mathtt{C},\mathtt{w}),\ \text{return } \mathtt{T} \text{ if } \mathtt{C} \text{ is satisfied in } \mathtt{w};\\ &\quad \text{otherwise, return } \mathtt{U}.\end{aligned}$$

DPLL can be executed by a different kind of constraint specialist. Storing a proposition performs elaboration

$$\begin{aligned}\mathtt{Store}(\mathtt{P},\mathtt{TV}): \mathtt{P} \text{ and } \mathtt{TV} \text{ are added as a clause to } \mathtt{C};\\ \text{perform the DPLL elaboration step on } \mathtt{C}.\end{aligned}$$

`ModifyFM(P, Q)`: If `C` is satisfied in the world w of `P`, then add `Satisfied(C, E, w)` to `Q`. Otherwise, randomly choose a proposition `P` *whose truth value remains uncertain* and put it, and its negation, at the front of `Q`. (It is common to improve DPLL with variable selection heuristics. There is no obstacle to incorporating these heuristics into `ModifyFM`.)

$$\begin{aligned}&\mathtt{OpinionOn}(\mathtt{P}): \text{If } \mathtt{P} \text{ is of the form}\\ &\mathtt{Satisfied}(\mathtt{C},\mathtt{w}),\ \text{return } \mathtt{T} \text{ if } \mathtt{C} \text{ is satisfied in } \mathtt{w};\\ &\quad \text{otherwise, return } \mathtt{U}.\end{aligned}$$

A system with repeated experience in similar environments will have made many inferences and solved many problems. One can speed it up using a form of CBR. A "case specialist" remembers the previous states of the worlds and the relations among objects in those situations. In a new situation, it is capable of finding similarities with the past and suggesting cases relevant to the present.

More specifically

$$\mathtt{Store}(\mathtt{P},\mathtt{TV}).\ (\mathtt{P},\mathtt{TV}) \text{ is added to memory.}$$

`ModifyFM(P, Q)`. If the *truth value of* `P` *is uncertain* and it is involved in a structure (i.e., a set of related propositions) that is highly similar (above some threshold) to a proposition `P1` in a previously encountered structure, then transform the old case into propositions that use the objects and times in the current situation, and put those propositions on the queue.

The three questions raised by this approach to CBR involve how the relevant features to a case are focused on, how cases are stored in memory, and how similarity between cases is computed. The last two questions are beyond the scope of this paper because our goal here is to provide a framework for integrating computational methods and not determining which specific methods are worth integrating. As for the first question, some of the features of a case must be focused on because they were perceived or inferred by other specialists. These will cue a case. The `ModifyFM` procedure described earlier would then have the result of adding focus for the rest of the features.

As an example of integration using focus traces, consider that CBR and WalkSAT integrate easily. WalkSAT proceeds as it normally would, flipping the truth value of one proposition at a time and focusing on it. When the case specialist finds a situation in the past that is sufficiently similar to this one, it essentially flips several truth values at once, hopefully getting WalkSAT much closer to a solution. The brittleness of CBR is ameliorated by the fact that whatever inconsistencies there are between the past case and the current situation can be resolved by continued search by WalkSAT.

Finally, for environments that are changing and/or are perceptible only through noisy sensors, every step of CBR and WalkSAT should be influenced by the information from these sensors.

Adding a "perception specialist" achieves this

$$\begin{aligned}&\mathtt{ModifyFM}(\mathtt{P},\mathtt{Q}).\ \text{If sensors perceive that a proposition}\\ &\quad \mathtt{X} \text{ has } changed\ its\ truth\ value,\ \text{put } \mathtt{X}\\ &\quad \text{at the beginning of } \mathtt{Q}.\\ &\mathtt{OpinionOn}(\mathtt{P}).\ \text{Return the truth value that } \mathtt{P} \text{ had}\\ &\quad \text{the last time it perceived it or}\\ &\quad \mathtt{U} \text{ if you have never perceived it.}\end{aligned}$$

Thus, during any attention fixation caused by the constraint or case specialist, if new information about a proposition `P` is perceived, the perception specialist will ask the focus manger to focus on it. When Polyscheme does focus on `P`, the perception specialist will take the appropriate stance on it. The case and constraint specialists will then learn about it through their `Store()` procedures and incorporate the information into their case matching and search, respectively.

These examples illustrate that algorithms from very different computational frameworks can be integrated using the same "computational building blocks," i.e., the focus of attention of a set of modules.

V. Benefits of Focus Trace Integration

We will describe how Polyscheme enables the best characteristics of multiple diverse computational methods to be combined by describing a working mobile robot [14] controlled by Polyscheme. The robot's goal (shown in Fig. 2) was to keep track of and follow another robot as it moved about and occasionally became occluded by other objects. This task requires many of the characteristics of intelligent systems we described in the introduction. The reasoning and planning problems were difficult. Because the actions of the robots had side effects and because there was incomplete information (because of occlusion), traditional planners could not be used. Formulating the physical constraints on objects (e.g., they do not pass through

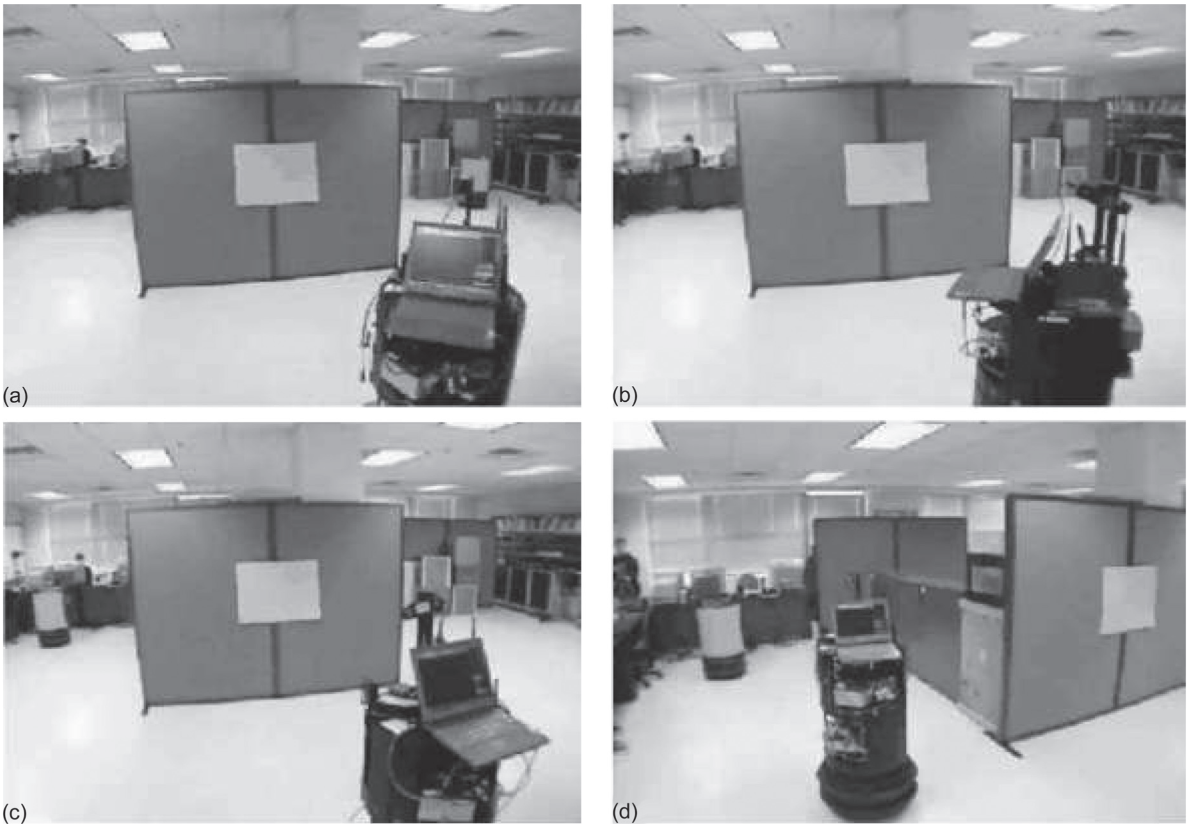

Fig. 2. Robot tracking problem. The robot in the foreground of (a) is tasked to track the robot in the distant right of (a). The tracked robot (b) disappears behind an occluder, and a robot with the exact appearance (c) emerges from the left of the occluder. Polyscheme infers that the two are identical and moves the tracking robot toward the visible robot. When (d) it sees an obstacle to robot motion behind the occluder, it infers that the two robots are different.

each other, gravity, etc.) in this domain in a SAT solver was prohibitive. Memory demands grew exponentially with spatial resolution, exhausting the memory of typical desktop computers with grid sizes around $5 \times 5 \times 5$. Furthermore, although operating in an environment while taking into account physical laws and potentially occluded objects is well beyond the reach of pure reactive systems, the positive characteristics of these systems were required. The environment was constantly changing; thus, the robot had incomplete information, and it therefore needed to be reactive and flexible enough to incorporate new information from its sensors into its planning and inference.

We briefly describe the Polyscheme system that controlled the robot. The focus manager was a modification of the queue scheme described earlier. The main modification was that propositions in the queue were associated with "satisfaction conditions" that would cause the proposition to be removed from the queue when they were met. For example, if proposition `P1` was put on the queue to help infer if `P2` was true, then if the truth of `P2` is determined, there is no longer a need to focus on `P1`, and it is removed. The specialists included a perception specialist that detected the location and type of objects in the environment. A physical constraint specialist kept track of physical constraints, and a path specialist included a library of "path scripts" that described paths that robots typically take.

We can now use this robot to help describe how Polyscheme enables systems that exhibit the characteristics of algorithms based on diverse computational formalisms.

A) Generality and Flexibility: Methods such as local search, backtracking search, and stochastic simulation make inferences, find plans, or solve constraints in a very wide variety of domains. This makes them general and flexible in that, when a situation changes and is formulated for these algorithms, they will deal with them accordingly. We have already described how to implement such algorithms within Polyscheme. In our robot, these kinds of algorithms were used to maintain the physical constraints described before. The benefit of doing so is that the system can also exhibit the following characteristics, which are not often exhibited in pure versions of these general algorithms when operating on many classes of problems.

B) Speed: General algorithms tend to be slow on larger problems because the state spaces they explore grow very quickly as a problem grows. "Structured" reasoning and planning algorithms based on frames or scripts do not have this problem since they do not generally search state spaces but instead make inferences or solve problems using large structured representations (i.e., frames, scripts, or cases). When these algorithms are implemented in Polyscheme together with more general and flexible algorithms, the best characteristics of each can be exhibited in the same system. In our robot, Polyscheme could find a continuous path between two sightings of a tracked object more quickly than pure search because of its library of path scripts. When a script was retrieved that did not completely match the existing situation, the constraint system would detect this contradiction, and this would initiate a search for a model for changes to the script that would be more consistent with the situation. While full search was always available, the script retrieval effectively meant that the search began in a state much closer to the correct model of the world. Thus, the speed of structured approaches was combined with the generality and flexibility of search-based methods.

C) Reactivity: Environments whose states change and which are sensed through imperfect sensors require systems to be able to quickly update their plans and inferences upon new information. Reactivity can be achieved using Polyscheme by mandating each focus of attention to be quick and by including sensor specialists. This guarantees that little inference will happen before new information is sensed. When our robot moved to a different location and saw objects that were formerly occluded, the perception specialist requested that all the other specialists focus on this information immediately. When they did, the constraint and path specialists were able to update their inferences and plans accordingly.

The ability of the robot to engage in complex reasoning and planning while moving about in a changing world demonstrates that creating hybrids of algorithms in Polyscheme can enable a combination of characteristics not possible given the existing individual methods alone.

VI. Evaluations and Implemented Systems

We used Polyscheme to create several systems that illustrate the benefits of this approach to hybrid algorithm execution. These benefits can be measured quantitatively, and they can be illustrated by systems that provide qualitatively new functionality.

A. Quantitative Evaluations

To confirm that Polyscheme's integration of multiple data structures and algorithms to constrain inference could lead to computational speedups, we performed quantitative evaluations on path-planning and spatial-reasoning problems.

1) Path Planning: Finding a path through a graph is an important problem in several fields, particularly robot motion planning. It is common to discretize a continuous space into a graph and use an algorithm such as A^* [15] to find the optimal path through the graph. This approach requires several assumptions to be made that cannot be obtained in many real-world environments. The most important of these assumptions involves change. For example, these algorithms typically do not enable planning to account for a change in a robot's abilities. If a robot that can fit through a door picks up and carries a large object, it may no longer be able to move through the door. Thus, the link between the regions that the door connects must be severed in the graph. However, most path-planning algorithms assume a fixed graph. Another example of change that traditional motion planning algorithms do not account for involves changes in the environment that do not involve the robot. For example, if a door closes automatically during certain temporal intervals, then the graph representing the connectivity of the regions must change. (Even path-planning algorithms, such as Ariadne [16], that use multiple kinds of algorithms to improve search do not deal with change through time of this sort.) For reasons such as these, dedicated motion planning algorithms cannot work in many real-world situations.

One solution to addressing the problem of action effects and change is to formulate path planning as a weighted SAT problem. Side effects and changes of state would be easy to formulate as SAT constrains. However, to represent that the states of objects can change over time, SAT encodings of such problems must include a copy of each state variable for every time step. For example, it is not sufficient to have a $door17Open$ variable. One must have $door17OpenAtTime1$, $door17OpenAtTime2$, etc., variables. Thus, the size of the required SAT formulation would be very large for situations that extend over many time steps. However, the execution time of SAT algorithms, as will be illustrated hereinafter, grows exponentially with the number of variables. Thus, for problems with many time steps, SAT encodings are not an efficient means of planning paths.

One solution to this problem is to introduce temporal intervals. Rather than write that $door17$ is open at times 3, 4, 5, 6, 7, 8, 9, 10, and 11, one can write that $door17$ is open over the interval (3, 11). However, SAT encodings would require that every possible interval be represented. Thus, for example, a domain with 100 time points entails tens of thousands of temporal intervals with those times as end points. Furthermore, this formulation would not reduce the search space. To move from A to B, the search algorithm would have to consider the world where the motion occurred at time 1, the world where it occurred at time 2, and so on, even though, in most cases, the specific time is irrelevant. The root cause of the problem is that the SAT formulation requires all constraints to be grounded propositionally and that it cannot reason over indefinite objects. If they could, then they could plan under the assumption, e.g., that the robot moved from A to B at an "indefinite" time t and the only reason about the specific bounds on t if they become relevant.

To enable reasoning over new objects, we created a system for "generative" SAT (GenSAT) solving called GenDPLL [17]. It implements DPLL in the manner outlined in Section III. However, rather than encoding constraints as propositional SAT problems, it encoded them using first-order constraints. These constraints were stored in a "formula specialist" that used a rule matching algorithm to perform the DPLL elaboration step.

This enables intervals to be represented as "indefinite" times that are constrained between two initially unknown points. For example, the following constraint encodes that, if an object moves from being in A at time a to being in nonadjacent B at b, it must have traveled to an intermediate adjacent location at some time point

$$Loc(?x, ?p_1, ?t_1) \wedge Loc(?x, ?p_2, ?t_2) \wedge \neg\, Same(?p_1, ?p_2)$$
$$\rightarrow Adjacent(?p_1, ?p_x) \wedge Meets(?t_1, ?t_x), ?p_x, ?t_x.$$

Thus, indefinite times eliminate the need to consider all the possible end points of t, significantly reducing the search space.

We tested the increasing efficiency so enabled in a robot path-planning domain. We presented Polyscheme and planners based on modern weighted SAT solvers a 2-D grid and two end

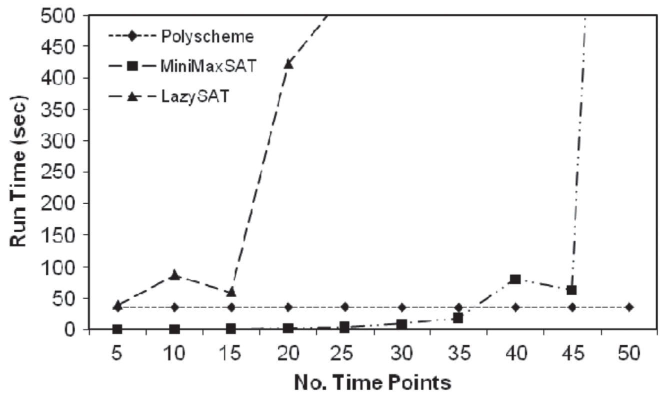

Fig. 3. Performance of Polyscheme, MiniMaxSat, and LazySat on a path-planning problem.

points. The grid contained obstacles whose locations changed over time and, thus, for the reason mentioned before, required the capability to reason about change. The goal of these systems was to find the shortest possible path between points. The fact that objects could change properties and locations made many common path planners unusable for this domain and, for the reasons mentioned earlier in the section, made this a very hard search problem for conventional SAT algorithms. Because we desired the ability to include constraints that were not purely about paths, we could not simply consider time as a third dimension and use a 3-D path planner.

We compared Polyscheme's performance against LazySAT [18] and MiniMaxSAT [19]. LazySAT was used because it is the only weighted MaxSAT solver that lazily instantiates constraints. LazySAT uses a WalkSAT-like [5] local-search algorithm. Since such algorithms, in many cases, perform worse than systematic solvers, we also evaluated MiniMaxSAT's performance. MiniMaxSAT is a weighted SAT solver based on MiniSAT, which is, in turn, an extension of MiniSAT. MiniSAT, the winner of the 2006 SAT Competition, is widely regarded as one of the best available SAT solvers.

As Fig. 3 shows, the performance of GenDPLL was approximately constant in the number of time points and performed faster than LazySAT on problems with more than approximately five time points and MiniMaxSAT with more than 35 time points. The graph reflects CPU times and thus demonstrates that, even though Polyscheme involves significantly more computation for any step in the search, its ability to use multiple data structures and algorithms to prune search is often able to significantly compensate for this overhead.

Averages of ten runs for each number of time steps were used in forming the graph in order to average away changes in performance due to background processes on our computer or the specific problem instances. Since LazySAT is not guaranteed to halt with an optimal solution, the times displayed indicate the speed with which it found a path that did not break hard constraints (e.g., that robots cannot pass through solid objects). Although LazySAT lazily instantiates constraints, it does not reason over unknown objects and thus achieves neither reductions in the number of explored models nor the consequent speedups that are afforded by the GenSAT interval formulation. In the indefinite interval formulation, merely adding time steps does not change the number of models that need to be considered.

These results demonstrate that, by using Polyscheme to create a hybrid of a SAT-solving algorithm (GenDPLL) and a rule matcher, problems which were not solvable by these or other methods alone can be solved efficiently.

2) Spatial Reasoning: Domains that involve spatial relations can pose a problem for inference algorithms because of the number of points they involve. A 100×100 grid, for example, contains 10 000 points. If one is uncertain about the location of A and only knows that point B is within ten cells away from B, then there are on the order of one million possible configurations of A and B in that grid that are consistent with this knowledge.

There are many "diagrammatic" reasoning systems that enable reasoning with such spatial constraints. They tend to be more efficient than, for example, SAT solvers on such problems in part because diagrams more compactly represent spatial relations. Existing diagrammatic reasoning systems are however quite limited in their ability to also reason over nonspatial constraints, and no such systems offer the generality, soundness, and completeness of SAT solvers.

To address this problem, we created a hybrid system intended to provide the benefits of SAT methods while enabling considerably more efficient inference on problems with spatial relations. The system was able to take input in the same form as other SAT solvers, except that some of the constraints could involve (possibly metric) spatial relations. Examples include $Near(a, b, 10)$ ("a is within ten units of b), $Left(b, c)$, and $Above(a, c)$. Constraints could mix spatial and nonspatial relations so that it would be possible to represent a constraints to the effect of "Dogs on a leash are near the person holding the leash." Given such constraints as input, the system finds models that satisfy them, if they exist.

This system, called DPLL-S, was based on a weighted SAT solver similar to the one described in the last section. In addition, the system included a diagram component that kept track of spatial relations and used "possibility spaces" [20] to compactly represent relations. Thus, rather than needing to represent every specific possible location of an object, it could represent the region (i.e., possibility space) where the object could potentially be located. This representation would enable the diagram component to rule out many possible object locations very quickly. For example, if A is more than 70 units from B and C is within ten units of B, then, using standard grid algorithms, one can very quickly infer using possibility spaces that A is not near C. The only way to rule out that possibility using a standard SAT solver would be to search over potentially millions of possible configurations of A, B, and C. The diagram specialist can therefore dramatically reduce the number of possibilities explored during search.

To measure the impact of this hybridized approach, we tested the system on problem instances of varying size. Since a criticism of many diagrammatic reasoning systems has been that they are tested only on problems that suit them, we randomly generated problems. We then solved these problems using MiniMaxSAT and our hybrid system. On problems with smaller grid sizes, MiniMaxSAT outperformed the hybrid

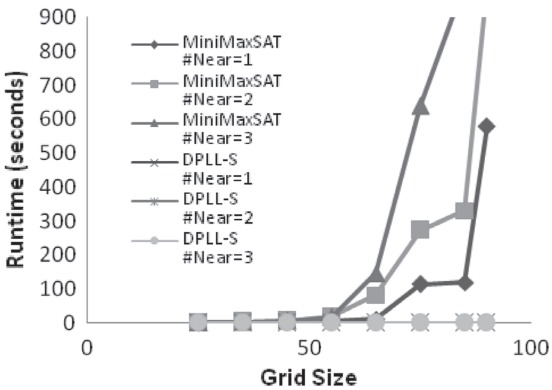

Fig. 4. Performance of DPLL-S versus MiniMaxSAT.

system. However, with increasing grid sizes, the benefits of reasoning with possibility spaces increased, and the hybrid system significantly surpassed MiniMaxSAT's performance. Fig. 4 shows these results. "$Near = n$" in the legend indicates that the problems run had the no $Near$ predicate had an argument greater than n. As n grows, there are generally more possible configurations that satisfy a constraint, and thus, performance deteriorated.

3) Computational Complexity of Search: The path-planning and spatial-reasoning problems we have been discussing have a high degree of computational complexity. We make no claim that our approach somehow changes the complexity class of these problems nor do we claim to have surmounted the "no-free lunch" [21] theorems, which hold that "any two algorithms are equivalent when their performances are averaged across all possible problems" [22]. It is however very common for algorithmic improvements that do not change complexity characteristics to nevertheless make an approach tractable on problems that had been impossible before. Innovations such as stochastic local search [23] and clause learning [6] have, for example, made SAT solvers usable on a wide range of new problems without altering the fact that SAT solving is a nondeterministic polynomial time complete problem. Although the technical details are quite different, Polyscheme follows in this tradition.

Furthermore, the performance gains mentioned here were specific to domains with spatial and temporal relations. This is to be expected since the gains were achieved by hybridizing search with specialized spatial and temporal reasoning methods. While the specificity of these improvements is a limitation of these systems, there is a wide body of research in cognitive science [24], indicating that reasoning in many domains can be mapped onto reasoning about a set of relations such as time and space. Confirming the potential breadth of application of such specializations is a topic for future research.

Finally, Polyscheme involves a significant amount of overhead that requires considerably more computations per state explored during search. This is confirmed by our quantitative evaluations. However, in the case of path planning, we showed that the new problem formulation that Polyscheme enables significantly reduces the search space and, hence, the CPU time needed, as the problems involved larger time frames. The spatial-reasoning search showed how a diagram module could significantly reduce the number of states explored. Since all our evaluations measured CPU time, they demonstrate that, in many cases, hybridized search can outperform deeper search using a single algorithm.

B. Systems With New Functionality

Although quantitative evaluations help measure and precisely characterize specific advances that are enabled, our ultimate goal in this paper is to enable intelligent systems with functionality that formerly had not been straightforward to create or which solved problems that could not be solved (in theory, not just efficiently) in other frameworks.

The principal way to demonstrate that Polyscheme can enable this is to actually build such systems. The robot from the last section is a prime example. It can react to changes while making inferences and finding plans that purely reactive systems cannot and that conventional inference and reasoning algorithms achieve only (as we described in the SAT example given earlier) on very small problem scales.

A heterogeneous database retrieval system [25] implemented in Polyscheme also illustrates its benefits. This system makes sound and complete inferences over heterogeneous sources of computation and information. It implements resolution theorem proving using common functions, such as forward inference, subgoaling, and identity. Each of these operations is implemented in specialists based on representations, such as neural networks, production rules, geospatial coordinates, and relational databases. If these modules meet certain conditions (which, in practice, are easy to confirm), the total system's inference is sound and complete. This system demonstrates how hybridizing algorithms provides both the benefits of logic programming (provable soundness and completeness) and the efficiency of special representations (such as those from neural networks and relational databases).

Finally, the GenSAT language and GenDPLL algorithm we mentioned in the motion planning section overcome some severe difficulties that arise in domains with unknown objects. Languages, such as GenSAT, that license the inference of objects that are unknown before inference can lead to models with infinite numbers of objects. For example, a constraint to the effect that "all mammals have a mother" and "a person cannot be their own ancestor" requires a model with infinite numbers of ancestors for any particular mammal. As another example of finite theories with infinite models, many context-free grammars license infinite numbers of derivations. Because traditional SAT solvers require all objects to be known in advance—this is true even of lazy SAT solvers such as LazySAT—they cannot solve many problems in such domains. In [17], we prove that hybridizing rule matching with weighted constraint solving in GenDPLL enables models to be found of GenSAT theories even in many conditions where there are infinitely many models with infinite numbers of unknown objects. In [26], we show how to use GenDPLL to provide parses for probabilistic context-free grammars (PCFGs), even in cases where a grammar has infinite numbers of derivations. Since GenSAT can encode an extremely wide variety of constraints, this result enables linguistic knowledge (in the form of PCFG constraints) and nonlinguistic knowledge to jointly constrain interpretation during parsing. The absence of such an ability has been a serious obstacle to the use of context to disambiguate language.

The heterogeneous database retrieval and GenSAT results had been beyond the theoretical reach of existing approaches. No approach had previously enabled sound and complete answers to queries over information in such a wide variety of formalisms, and no other approach had enabled reasoning over constraints as general and flexible as those in SAT to be jointly reasoned over with grammars that had infinite derivations. These results thus demonstrate that executing hybrid algorithm through a focus of attention in Polyscheme not only can speed inference but also enable inference in situations that had heretofore been theoretically intractable.

VII. Other Approaches to Integration

Most approaches to integrating algorithms and/or their characteristics into a single system have taken a reductive, modular, or "fixed" hybrid approach. Reductive approaches tend to implement an algorithm or solve problems by a reduction to another approach. This often consigns such systems to the limitations of the computational formalism or method being reduced to. For example, reducing a first-order probabilistic logic reasoning problem to a graphical model belief propagation problem [27] means that one is still limited by the propositional representation and scalability characteristics of belief propagation inference algorithms. Modular architectures for integration tend to enable modules based on different data structures and algorithms to communicate and/or cooperate. Normally, the only way to add an algorithm to such a system is to add a module based on that algorithm. Polyscheme enables this kind of integration, but it also enables algorithms to be executed through the focus of attention. This allows every single step of every algorithm to be executed using many data structures and algorithms and thus enables a much more thorough integration of algorithms. Fixed hybrid approaches, such as Clarion [28] and ACT-R [29], [30], create hybrids between a few specific algorithms. Both include production systems that use reinforcement-learning mechanisms for conflict resolution but do not enable such close interaction between other algorithms implemented in those systems.

Polyscheme differs from many cognitive architectures in being primarily inferential and not procedural. Most architectures, e.g., Icarus [31], Soar [11], Epic [32], and ACT-R, choose an action at every step. These actions are actual physical actions or operations on data structures in memory. In Polyscheme, at every time step, specialists do not take or propose actions but instead take stances on the truth value of propositions and suggest propositions to attend to. A consequence is that Polyscheme includes mechanisms for communication and sharing information about the truth of propositions that make it much easier to implement many reasoning and inference algorithms.

A focus of attention is a very important part of ACT-R and Rao's work [33] on visual routines. However, in ACT-R, the focus of attention is not used to implement algorithms and, in neither case, is it feasible (because of the absence of mechanisms involving truth values and alternate worlds) to implement and integrate reasoning and inference algorithms using this sort of focus of attention mechanism.

Self-adaptive system architectures (e.g., [34] and [35]) share some features of this approach. They often contain multiple algorithms based on different methods. Furthermore, these architectures often contain some form of self-monitoring that detects problems among system components and reacts accordingly. This is in some ways similar to the role of choosing focus to deal with metacognitive problems in Polyscheme. However, these systems typically do not contain a notion of a focus of attention that all modules process simultaneously, and they do not require all components to be able to simulate alternate worlds. Thus, while nothing prevents these systems from including components that implement inference algorithms inside the modules, they cannot implement these algorithms as the result of a guided focus of attention that involves all modules in every step of inference.

Polyscheme's focus of attention is superficially reminiscent of blackboard systems (e.g., [36]) insofar as it is a shared-entity multiple-module access. However, the two have numerous and profound differences: The focus of attention in Polyscheme is tiny; Polyscheme has no shared memory among modules while blackboards are such shared memory; Polyscheme modules can have arbitrarily large, varied, and persistent memories while, in many blackboard systems, the blackboard is the main form of memory; blackboard systems have no straightforward way of exploring alternate states of the world; and, unlike the focus of attention in Polyscheme, modules in blackboard systems generally do not synchronize their operation. Most importantly, the lack of shared memory frees modules in Polyscheme to use a much more diverse array of representational formalisms.

Finally, like the current approach, Beal [37] proposes that many solutions to human intelligence arise from the interaction of specialized processors and explore the value of a focus of attention [38]. While it does not implement inference algorithms using a focus of attention and simulated worlds, it does provide a means of learning interactions among modules. In Polyscheme, at present, these interactions are established by the system designers, although it is likely that additional benefits would result by using such methods to influence the interaction of specialists in Polyscheme.

VIII. Conclusion

The goal of this paper has been a framework for creating a single system that can exhibit the best characteristics of algorithms based on different computational formalisms. Our approach has been to create a cognitive architecture called Polyscheme that can execute hybrids of these algorithms. Polyscheme enables two kinds of hybrids. First, a Polyscheme system can include modules based on arbitrarily different algorithms and data structures as long as they implement the common functions. Second, and most originally, Polyscheme can execute algorithms by guiding the "focus of attention" of these modules. Systems created using Polyscheme demonstrate that integration through a focus of attention enables systems that can make inferences and solve problems in situations beyond the reach of existing individual computational methods.

References

[1] M. L. Minsky, "A framework for representing knowledge," in *The Psychology of Computer Vision*, P. H. Winston, Ed. New York: McGraw-Hill, 1975.

[2] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals, and Understanding: An Inquiry Into Human Knowledge Structures*. Hillsdale, NJ: Lawrence Erlbaum, 1977.

[3] J. Kolodner, *Case-Based Reasoning*. Menlo Park, CA: Morgan Kaufman, 1993.

[4] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-6, no. 6, pp. 721–741, Nov. 1984.

[5] B. Selman, H. Levesque, and D. Mitchell, "A new method for solving hard satisfiability problems," in *Proc. 10th Nat. Conf. Artif. Intell.*, 1992, pp. 440–446.

[6] N. Een and N. Sorensson, "MiniSat—A SAT solver with conflict-clause minimization," in *Proc. SAT Competition*, 2005. AQ8

[7] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. 39th Des. Autom. Conf.*, Las Vegas, NV, 2001, pp. 530–535.

[8] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962.

[9] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[10] M. I. Jordan and C. Bishop, "Neural networks," in *CRC Handbook of Computer Science*, A. B. Tucker, Ed. Boca Raton, FL: CRC Press, 1997.

[11] J. E. Laird, A. Newell, and P. S. Rosenbloom, "Soar: An architecture for general intelligence," *Artif. Intell.*, vol. 33, no. 1, pp. 1–64, Sep. 1987.

[12] N. L. Cassimatis, "Reasoning as cognitive self-regulation," in *Integrated Models of Cognitive Systems*, W. Gray, Ed. New York: Oxford Univ. Press, 2007.

[13] A. M. N. Cassimatis, "Reasoning as perceptual simulation," in *Cognitive Processing*, to be published. AQ9

[14] N. L. Cassimatis, J. G. Trafton, M. Bugajska, and A. C. Schultz, "Integrating cognition, perception, and action through mental simulation in robots," *Robot. Auton. Syst.*, vol. 49, no. 1/2, pp. 13–23, Nov. 2004.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[16] S. L. Epstein, "Pragmatic navigation: Reactivity, heuristics, and search," *Artif. Intell.*, vol. 100, no. 1/2, pp. 275–322, Apr. 1998.

[17] A. M. N. Cassimatis and P. Bignoli, "Inference with relational theories over infinite domains," in *Proc. FLAIRS*, 2009. AQ10

[18] P. Singla and P. Domingos, "Memory-efficient inference in relational domains," in *Proc. AAAI*, 2006, pp. 488–493.

[19] F. Heras, J. Larrosa, and A. Oliveras, "MiniMaxSAT: An efficient weighted Max-SAT solve," *J. Artif. Intell. Res.*, vol. 31, pp. 1–32, 2008.

[20] S. Wintermute and J. E. Laird, "Predicate projection in a bimodal spatial reasoning system," in *Proc. 22nd AAAI Conf. Artif. Intell.*, Vancouver, BC, Canada, 2007, pp. 1572–1577.

[21] D. H. Wolpert and W. G. Macready, *No Free Lunch Theorems for Search*. Santa Fe, NM: Santa Fe Inst., 1995.

[22] D. H. Wolpert and W. G. Macready, "Coevolutionary free lunches," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 721–735, Dec. 2005.

[23] J. Gu, "Efficient local search for very large-scale satisfiability problems," *SIGART Bull.*, vol. 3, no. 1, pp. 8–12, Jan. 1992.

[24] N. L. Cassimatis, "A cognitive substrate for human-level intelligence," *Artif. Intell. Mag.*, vol. 27, no. 2, pp. 45–56, Jul. 2006.

[25] N. L. Cassimatis, "A framework for answering queries using multiple representation and inference technique," in *Proc. 10th Int. Workshop Knowl. Representation Meets Databases*, 2003. AQ11

[26] A. Murugesan, N. L. Cassimatis, S. Dugas, and M. Bugajska, "Combining probabilistic context-free parsing with general inference," in *Proc. UAI*, Vancouver, BC, Canada, 2007.

[27] P. Domingos and M. Richardson, "Markov logic networks," *Mach. Learn.*, vol. 62, no. 1/2, pp. 107–136, Feb. 2006.

[28] R. Sun, "The CLARION cognitive architecture: Extending cognitive modeling to social simulation," in *Cognition and Multi-Agent Interaction*. New York: Cambridge Univ. Press, 2004.

[29] J. R. Anderson, "Human symbol manipulation within an integrated cognitive architecture," *Cogn. Sci.*, vol. 29, no. 3, pp. 313–341, May 2005.

[30] J. R. Anderson and C. Lebiere, *The Atomic Components of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1998.

[31] P. Langley and D. Choi, "A unified cognitive architecture for physical agents," in *Proc. 21st Nat. Conf. Artif. Intell.*, Boston, MA, 2006, pp. 1469–1474.

[32] D. Kieras and D. E. Meyer, "An overview of the EPIC architecture for cognition and performance with application to human–computer interaction," *Hum.-Comput. Interact.*, vol. 12, no. 4, pp. 291–438, Dec. 1997.

[33] S. Rao, "Visual routines and attention," in *Electrical Engineering and Computer Science*. Cambridge, MA: MIT Press, 1998, p. 86.

[34] H. E. Shrobe, R. Laddaga, R. Balzer, N. M. Goldman, D. Wile, M. Tallis, T. Hollebeek, and A. Egyed, "Self-adaptive systems for information survivability: PMOP and AWDRAT," in *Proc. SASO*, 2007, pp. 332–335.

[35] R. Laddaga, P. Robertson, and H. E. Shrobe, "Introduction to self-adaptive software: Applications," in *Proc. IWSAS*, 2001, pp. 1–5.

[36] B. Hayes-Roth, "A blackboard architecture for control," *Artif. Intell.*, vol. 26, no. 3, pp. 251–321, Jul. 1985.

[37] J. Beal, "Learning by learning to communicate," in *Electrical Engineering and Computer Science*. Cambridge, MA: MIT Press, 2007.

[38] J. Beal, "Shared focus of attention for heterogeneous agents," in *Proc. 7th Int. Conf. AAMAS*, 2008, pp. 1627–1630.

Nicholas Cassimatis, photograph and biography not available at the time of publication. AQ12

Perrin Bignoli, photograph and biography not available at the time of publication. AQ13

Magdalena Bugjaska, photograph and biography not available at the time of publication. AQ14

Scott Dugas, photograph and biography not available at the time of publication. AQ15

Unmesh Kurup, photograph and biography not available at the time of publication. AQ16

Arthi Murugesan, photograph and biography not available at the time of publication. AQ17

Paul Bello, photograph and biography not available at the time of publication. AQ18

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

AQ1 = Please validate the address and postal code provided for “Rensselaer Polytechnic Institute.”
AQ2 = Please validate the address and postal code provided for “Office of Naval Research.”
AQ3 = “Davis-Putnam-Logemann-Loveland” was taken as the expanded form of the acronym “DPLL.” Please check if appropriate.
AQ4 = “Case-based reasoning” was taken as the expanded form of the acronym “CBR.” Please check if correct.
AQ5 = “Rule specialist infers that B...” was changed to “rule specialist infers that B is true.” Please check if appropriate.
AQ6 = The acronym “NP” was defined as “nondeterministic polynomial time.” Please check if correct.
AQ7 = “Probabilistic context-free grammar” was taken as the expanded form of the acronym “PCFG.” Please check if appropriate.
AQ8 = Please provide page range in Ref. [6].
AQ9 = Please provide publication update in Ref. [13].
AQ10 = Please provide page range in Ref. [17].
AQ11 = Please provide page range in Ref. [25].
AQ12 = Please provide photograph and biography of the author “Nicholas Cassimatis.”
AQ13 = Please provide photograph and biography of the author “Perrin Bignoli.”
AQ14 = Please provide photograph and biography of the author “Magdalena Bugjaska.”
AQ15 = Please provide photograph and biography of the author “Scott Dugas.”
AQ16 = Please provide photograph and biography of the author “Unmesh Kurup.”
AQ17 = Please provide photograph and biography of the author “Arthi Murugesan.”
AQ18 = Please provide photograph and biography of the author “Paul Bello.”

END OF ALL QUERIES