Autonomous Robots, 5, 1–16 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Grounding Mundane Inference in Perception

IAN DOUGLAS HORSWILL
ian@cs.nwu.edu
The Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue
Evanston, IL 60201

Received June 1, 1997; Revised October 6, 1997

Editors: George A. Bekey

Abstract. We describe a uniform technique for representing both sensory data and the attentional state of an agent using a subset of modal logic with indexicals. The resulting representation maps naturally into feed-forward parallel networks or can be implemented on stock hardware using bit-mask instructions. The representation has "circuit-semantics" [34, 32], but can efficiently represent propositions containing modals, unary predicates, and functions. We describe an example using Kludge, a vision-based mobile robot programmed to perform simple natural language instructions involving fetching and following tasks.

1. Introduction

Suppose you want to program a robot to accept simple instructions like "bring the green ball here." A likely plan for solving this task might be:

1. Search for a green ball and track it visually as the ball and/or robot move
2. Drive to the ball
3. Grab the ball
4. Drive to the starting position
5. Release the ball

Executing such a plan robustly requires the robot to perform error detection and recovery. If the ball slips out of the gripper during transit, the robot needs to detect the situation and rerun steps 2 and 3. Unfortunately, the plan doesn't say that the ball needs to stay in the gripper, just that the robot should perform a grasp at a certain point in the plan.

This paper addresses two problems in autonomous agency. The first is the problem of making mundane inference and problem solving processes, such as determining how to deliver a ball, run fast enough that they are effectively instantaneous. If the problem solver runs fast enough, then error detection and recovery are easy: we just keep rerunning the problem solver from scratch. The second problem is grounding the problem solver's inferences in continually updated sensor information so that decisions about what to do next change in real time as sensor readings change.

I will argue that these two problems are related. I will present an architecture that allows a useful subset of modal logic to be compiled into fast feed-forward networks. The networks both drive and are driven by a modern active vision system on a real mobile robot (figure 1). The architecture allows designers to build systems with the performance characteristics of behavior-based systems [28], while simultaneously providing much of the generativity of traditional symbolic problem solvers.

1.1. Agent architectures

Much of the work in this decade on agent architectures has been concerned with the problem of merging the strengths of reactive and deliberative systems (see for example, [18, 17]).[1] A pure reactive system contains no internal state (memory) and consists of a set of task-specific, preprogrammed rules for firing actions based on immediate sensory input. The rules are typically implemented using some kind of parallel network whose inputs are driven by sensors and whose outputs drive the effectors. Purely deliberative systems (planners) compute in advance, and commit to, a complete series of actions intended to achieve the goal. The series of actions (the plan) is typically computed from the goal using a combination of search and simulation.

Both approaches have obvious flaws. It is easy to construct examples of non-Markovian environments in which a reactive system's lack of internal state will get it into trouble. The task-specificity of their rules also raises questions about scaling and generativity. On the other side, deliberative systems typically require exponential search, making them slow for simple problems and unusable for complex ones. Another issue is that purely deliberative systems commit to a plan in its entirety. The only way they can respond to unexpected contingencies is for the plan to fail entirely, triggering a restart of the planner. However, it can be difficult for the executive running the plan to know how to judge when the plan has failed, since the plan contains only the actions to perform, not their rationale. In a simple planner/executive architecture, all the domain knowledge is in the planner, but understanding whether a plan has failed requires at least as much knowledge as formulating it in the first place.

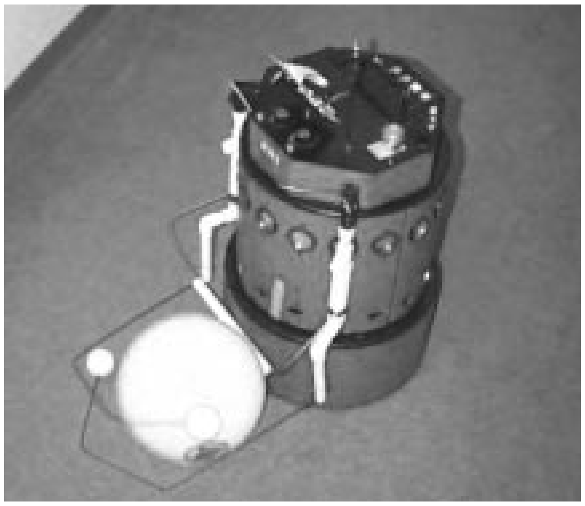
Fig. 1. Kludge the robot during a delivery task. Its job is to search for, approach, and grasp a ball of specified color and to deliver it to a person wearing a specified color or to a designated point in space. It must quickly recover from problems such as the ball slipping out of its mandibles or the ball being momentarily occluded.

Clearly, one wants a system that combines the strengths of reactive and deliberative systems. The most common approach to this has been to build a hybrid system incorporating reactive and deliberative systems as components, typically in a three-tiered architecture: a planner computes plans from goals, an executive sequences them, and a set of reactive systems implement the low level symbolic actions (see [3] for an extensive survey). One common argument for such an architecture is that deliberation will always be slower than reaction, so any division of labor between the two will have to allocate fast time-scale activities to the reactive system and slower time-scale activities to deliberative systems [5].

The other major approach is to select special cases of planning that can be mapped efficiently into parallel hardware. The networks then compute the same input/output mapping as a planner, but run in bounded time. The earliest example of this is Rosenschein and Kaelbling's system, which compiles propositional logic axioms into sequential circuits [34]. Kaelbling's GAPPS system [23] used goal regression to compile a (propositional) planner-like formalism into sequential circuits. Mataric implemented a Dijkstra-like shortest path finder using spreading activation in a behavior-based system [29]. Maes' behavior network system computed an approximation to propositional STRIPS planning using spreading activation [27].

The use of propositional logic (logic without variables or predicate/argument structure) is severely limiting, however, and there have been a few attempts to rectify it. Nilsson's TRT system [32] handles predicate/argument structure by incrementally growing a propositional network as new combinations of arguments are encountered. Agre and Chapman's *deictic representation* [2, 1] is an explicit attempt to overcome the limitations of propositional representations in which variable binding is performed in the perceptual system rather than the reasoning system.

1.2. Representation languages, complexity, and parallelizability

The problem with hybrid systems is complexity. Technically, planning isn't slow, it's *complex.* Planning systems typically take exponential time to compute their plans. If we imagine a planner that takes time exponential in the length of the completed plan, then while it might take 100 seconds to compute a 30 step plan, 3 seconds to compute a 25 step plan, and 3 microseconds to compute a 5 step plan, it would take 28 hours to compute a 40 step plan.[2]

The complexity problems of the parallel hybrids have more to do with hardware than time. Ultimately, the problem with the parallel hybrids is that they have no way of doing variable binding and so they are effectively limited to representations that are isomorphic to propositional logic rather than predicate (e.g. first order) logic. That means that instead of having a single node in the system that represents $on(A, B)$, they need to use separate nodes for each possible value of A and B. In the case of Maes' solution for the blocks world [27], that ends up meaning $O(n^3)$ nodes in the system and $O(n^4)$ wires connecting them. In Hasegawa et al.'s dialog system [16], a separate node is used for every possible sentence that can be make by either speaker.

There is a deep relationship between the expressiveness of a representation language and the computational complexity of its inference problem (see, for example, [24]). Propositional inference is quite tractable but so limited that one is forced to do things like represent all ground-instances of $on(A, B)$ separately. On the other hand, inference in first-order predicate logic is Turing-complete and so wildly intractable. If we remove certain features like term expressions from the language (so we can say $related(X, Y)$ but not $related(brother(X), X)$), we get the language DATALOG, whose inference problem is P-complete [37], meaning it is tractable, but so far as we know, unparallelizable.

In fact, parallelizing inference is extremely difficult. Even simple problems like unification, which are ubiquitous in AI systems, are P-complete and so thought to be inherently serial [10].

1.3. Perception systems

Contemporary active sensing systems typically have attentional components that must be controlled appropriately for the current task. For example, many active vision systems have the ability to track a small number of objects (typically one) [22, 8, 15, 13, 9, 26, 21]. Attentive tracking is necessary because it is impractical to completely analyze a stream of images — find and label all objects in each image, then match the identities of the objects between images — which would amount to tracking every possible object at once. Instead, tracking systems "latch on" to a small number of objects and continuously report their characteristics over time. For example, a disparity-based tracker such as [8] reports the image-plane coordinates of the object being tracked, as well as its disparity (a measure of distance). Trackers generally provide some way of initially choosing an object based on its visual properties [33, 20].

Control of modern perceptual systems therefore involves dynamic allocation of scarce sensory resources to task-relevant objects. To know the distance of an object, the system must first allocate a tracker to the object. Unfortunately, current reasoning systems have no way of directing these resources. They require all knowledge to be prestored in a symbolic database. This leaves the perceptual system in the position of having to guess what information might be useful to the reasoner.

Allocating perceptual resources intelligently requires that the agent have at least limited ability to represent its own attentional state and to reason about that state's relation to its present goals. This requires care, however, since the allocation of attention cannot depend on gathering information that would itself require allocating attention (on penalty of infinite regress).

In practice, symbolic architectures typically treat the allocation of a tracker or the checking of its output as a discrete action that is generated by the problem solver based on *ad-hoc* rules. In addition to being an added burden on the programmer, such rules leave open the possibility that the problem solver will forget to check something. Thus robots that perform delivery tasks such as our ball delivery task either have specific hand-coded rules to regularly check if the ball has fallen out of the gripper or do not check at all.

In theory, one could imagine driving a conventional serial reasoner with perceptual input by interfacing the perceptual system to the symbolic database so that the outputs of the perceptual systems are something like first-order logic assertions expressed as tree-like data structures. It is difficult to imagine how to do this in a domain-independent manner, however, since the perception system needs to know what information is relevant to the current task. Another issue is how to inform the reasoner about changes in the environment. A planner that is deep in a non-deterministic search may need to completely restart the search when the environment changes. Determining whether a given change is relevant and requires a restart of the search process is not an inviting problem. Another possibility is to build a perceptual system that emulates the symbolic database [20]. While useful, this approach still leaves open the problem of tracking environmental changes and updating stale inferences.

By far, the simplest way to ground an inference process is to choose a representation language whose inference rules can be compiled into parallel networks and then drive the inputs of those networks with the outputs of the perceptual system. Then the outputs of the inference rules will follow the perceptual system as its state changes. The issue, then, is to find the most useful representation language for which we can do this.

2. Role passing

We have developed a set of techniques for implementing limited inference and variable binding in otherwise behavior-based architectures. We call the class of architectures they encourage “role passing” architectures for reasons that should be clear in a moment. Role passing allows the use of more expressive representation languages without sacrificing parallelizability or the ability to ground inference in sensor data.

A role-passing system is composed of four basic types of components:

- A set of traditional sensory-motor *behaviors*, as in subsumption [6] or the Schema system [3].
- A set of specialized *short-term memory* systems. Each STM provides a way of representing a specific type of information about objects in the world. Internally, an STM is composed of a fixed pool of representations that can be *allocated* to hold information about a given object. Allocated representations are *tagged* with a specific tag (see below) that is used when looking up information about the object being represented. Once allocated and tagged, the representation is said to be *bound* to that tag. The set of possible tags is fixed in advance and is assumed to be relatively small.
- A set of *access ports* between behaviors and STMs. When a behavior drives the input of an access port with a tag, the STM drives the port’s output with the data from the representation bound to that tag.
- An *interference network* that receives data from the STMs, performs deductions using it, and generates control signals to drive the behaviors and STMs. This will be discussed more fully in section 3.

As with any parallel reactive system, these components should be thought of as pieces of hardware that continually recompute their outputs based on their current inputs, although in practice all our implementations run on standard serial computers.

Most of the communication within a role-passing system consists of passing tags from one system to another. When a behavior requires an object as a parameter, that object is specified by its tag. The behavior routes the tag to the appropriate access port and receives the data it needs about the object from the port’s STM.

The term “short-term memory” is somewhat misleading, since it implies a passive information store. Our use here is broader, however. In particular, we will group attentive perceptual systems, such as visual object trackers, into STMs. Such a collection of trackers is a memory insofar as it remembers to what objects the system is attending, and to what tags those objects are bound. It is, however, an "active" memory in the sense that it continually updates its own representations.

Our current implementation (see section 4), uses three STMs:

- A *visual tracking system* that can search for and track the positions of a set of designated objects. The system is composed of a set of individual trackers (3, in our implementation). Other components in the system can request that an object be tracked by specifying a color to search for and a tag to bind it to. The tracking system will then allocate a tracker to it, bind the tracker to the specified tag, and set it searching for the specified color.
- An *odometric tracking system* that can dead-reckon the positions of places the robot has been before. Other components can request that places be remembered by providing a tag to which to bind the place representation.
- A *description buffer* that stores information about the appearances of different objects. The present implementation stores statistics about the colors on an object's surface (e.g. mean RGB value, variances, etc). The descriptions can be used as input to the tracking system to initiate a search for an object with the specified statistics. Descriptions are bound by providing a set of statistics and a tag value to which to bind them. Unlike the other two STMs, the description buffer really is a passive information store.

Each of these STMs can potentially hold information about several objects. A given object can be represented in any, all, or none, of the STMs at a given time. Although we will focus on these three examples, many other types of STMs are possible. For example, Matarić's active map representation [29] could be considered an STM in our sense, as could marker-passing knowledge bases such as NETL [12].

The most speculative aspect of the architecture is the choice of tags. In role-passing systems, we have used linguistic role names (e.g. AGENT, PATIENT, SOURCE, DESTINATION) as tags. In other words, we assume that linguistic role names (and perhaps other features) are the keys by which short term memory is accessed. This may or may not be a good policy in the long run, but it has been useful so far. In any case, it is independent of the other architectural decisions, such as the use of tagging in general or the decision to break short-term memory up into specialized subsystems.

We can now sketch the solution of the delivery problem using role-passing. We bind the color of the object we want to deliver to the role PATIENT in the description buffer. We also bind our current location to the role DESTINATION in the odometric tracking system. We use an inference network that implements the following rules:

- Assert goal(in-hand(PATIENT))
- If in-hand(PATIENT),
  then assert goal(near(DEST))
- If in-hand(PATIENT) and near(DEST), then done
- If goal(in-hand(X)) and not(in-hand(X)), then assert goal(near(X))
- If goal(in-hand(X)) and not(in-hand(X)) and near(X), then activate the GRAB behavior with X as its argument
- If goal(near(X)),
  then assert goal(know(distance(X)))
  and goal(know(direction(X)))
- If goal(near(X)) and know(distance(X)) and know(direction(X)), then activate the DRIVETO behavior with the argument X.
- If goal(know(distance(X)))
  and not(know(distance(X)))
  and know(description(X)), then activate the visual search system with an argument of X.

and run the rules continuously and in parallel. At any given moment, the robot will drive to the destination iff current sensor data indicates that the ball is in the gripper. Similarly, it will attempt to acquire the ball iff sensor data indicates it is not in the gripper. Like parallel reactive systems, the robot will recover automatically from losing the ball. However, we can change the object it delivers or the destination it takes it to just by changing the initial role bindings. We do not have to duplicate rules or behaviors for each possible ball.

3. Grounding inference

Given the basic role-passing architecture, we can implement an inference network that supports limited forms of variable binding by treating roles as bound variables. Alternatively, we can think of roles as names with indexical reference. Since role names typically have functional significance (e.g. something bound to DESTINATION must be a place to which something can, has, or wants to go), role-passing can be thought of as a kind of indexical-functional (deictic) representation [2, 1].

3.1. Representing functions, predicates, and individuals

Peripheral systems, such as the behaviors and STMs are free to use whatever internal representations are convenient (images, sonar data, grid-based representations of space, etc.). However the inference network only uses three representations: short term memory keys (roles/tags), unary predicates, and scalar functions.

Predicates are represented by giving their values on every object bound in the system. Since there are only a finite number of roles, we can represent the value of a predicate as a bit-vector in which the ith bit being set means that the the predicate is true of the object bound to the ith role:

predicate	source	dest	patient	agent
near	T	F	F	T
see	F	T	T	F

As a slight abuse of terminology, we will call this the *extension* of the predicate. For convenience sake, we will use the same representation for roles when they are used as short-term memory keys: to refer to the object bound to the ith role, we use the bit-vector with only the ith bit set.

We can use a similar representation for real-valued functions over objects. A function is represented as a vector of numbers in which the ith value of the vector is the value of the function on the object bound to the ith role:

function	source	dest	patient	agent
distance	15	0	0	17
direction	14	87	35	-28

Since we can represent predicates and functions as small vectors, we can efficiently implement them on both von Neumann and parallel hardware. On von Neumann hardware, a function is a standard vector of memory cells and a predicate can be represented as a single memory word. Moreover, we can compute logical connectives with single machine instructions. A compound expression like:

$$P(x) \wedge (Q(x) \vee W(x))$$

can be evaluated by the code:

```
(logand p (logior q w))
```

(or `p&(q|v)` in C). The expression computes the bit-vector of the individuals for which the subexpression $P(x) \wedge (Q(x) \vee W(x))$ holds (assuming `p`, `q`, and `w` hold the bit-masks for P, Q, and W, respectively).

The representation is also conveniently implemented in a feed-forward network. Each predicate or function is implemented as a compact bus of wires holding the symbol's value for each indexical. Logical connectives are again implemented as bit-wise logic operations, but this time by constructing separate gates.

Thus, by limiting our representation to a small set of indexical names, and by limiting ourselves to single-place predicates and function symbols, we can efficiently represent logical expressions as fairly compact dependency networks. These networks are easily parallelized and easily updated in real-time.

3.2. Representing propositional attitudes

Thus far, we have no way of representing the distinction between $near(destination)$ being false, and $near(destination)$ being unknown. The distinction is critical for reasoning about attention.

The problem is easily solved by keeping a vector of valid bits in parallel with the truth bits of predicates and values of functions. Within the framework of logic, these valid bits are naturally interpreted as assertions about the agent's state of knowledge: if the truth bits encode the proposition $P(x)$, then the valid bits encode the proposition "know-whether $P(x)$", which we will write $\triangle(P(x))$:

$$\triangle(P(x)) \equiv K(P(x)) \vee K(\neg P(x))$$

$$\triangle(f(x)) \equiv \exists y.K(f(x) = y)$$

Since we ultimately want to build a problem solver, it is also useful to add a "goal" modality, so that we can express that $P(x)$ is a goal or even that $\triangle(P(x))$ is a goal. Thus the real representation of predicates is not just a bit-vector, but four bit-vectors: the predicate's extension, its valid bits, its goal bits, and its knowledge-goal bits. Functions are three bit-vectors and a scalar vector because functions are scalar-valued, not truth valued.

3.3. Derived functions and predicates

We can now define functions and predicates in terms of other functions and predicates. Given an axiom of the form

$$\forall x.P(x) \equiv \Phi(Q_1(x), ..., Q_n(x))$$

we can generate hardware or machine code to compute P from Q_i as described in section 3.1. We can also generate the axiom

$$\forall x.\triangle(Q_1(x)) \wedge ... \wedge \triangle(Q_n(x)) \Rightarrow \triangle(P(x))$$

which, again, is easily implemented as a bit-vector operation, this time on the valid bits.

At the cost of extra complexity, we can make this axiom stronger, at least for the standard logical connectives. For example, if we have that

$$\forall x.P(x) \equiv Q(x) \wedge W(x)$$

then we have that

$$\forall x.\triangle(P(x)) \equiv$$
$$(\triangle(Q(x)) \wedge \triangle(W(x))) \vee K(\neg Q(x)) \vee K(\neg W(x))$$

Goals are a more complicated manner. If we define $P(x)$ as $Q_1(x) \wedge ... \wedge Q_n(x)$, then we may often want to include the axioms:

$$\forall x.goal(P(x)) \Rightarrow goal(Q_1(x)) \wedge ... \wedge goal(Q_n(x))$$

$$\forall x.goal(\triangle(P(x)))$$
$$\Rightarrow goal(\triangle(Q_1(x))) \wedge ... \wedge goal(\triangle(Q_n(x)))$$

However, this can get us into trouble, since it may be important to solve one subgoal before the other. Thus while these axioms are useful defaults, they are not universally true. In our implementation, we allow conjunctions to solve their subgoals either in parallel, as above, or serially (see section 4).

3.4. Grounding primitive percepts

In order for our predicate/function representation to be useful, the perceptual systems must be able to generate it themselves. Fortunately, this is easy to do.

Recall that we have grouped perceptual systems together into STMs. Each STM contains a homogeneous pool of perceptual systems, such as visual object trackers or odometric trackers, or passive representations such as color descriptions. Each STM allocates perceptual systems (trackers, in this case) on demand and keeps track of the roles to which they are bound. The perceptual systems, for their part, measure various functions and predicates. For example, the visual object trackers might measure the predicate *is-moving* and the functions *distance* and *direction*. The STM can compute the complete extension of a predicate, such as *is-moving*, by forming a bit-vector of all the role tags of all the trackers that are reporting motion. It can compute the valid bits by forming a bit-vector of all the roles to which any tracker is bound.

In general, for an STM comprised of perceptual systems $T_1, ..., T_l$, each of which measures a set of predicates $P_1, ..., P_m$, and a set of functions $f_1, ..., f_n$), we can compute the extension and valid bits of P_i or f_i as:

$$\text{ext}(P_i) = \bigvee_{\{j|P(T_j)\}} \text{role}(T_j) \quad (1)$$

$$\text{ext}(\triangle(P_i)) = \bigvee_{j} \text{role}(T_j) \quad (2)$$

$$\mathrm{f_i}(r) = \begin{cases} f(T_j), & r \in \mathrm{role}(T_j) \\ \text{undefined}, & r \text{ not bound} \end{cases} \quad (3)$$

$$\mathrm{ext}(\triangle(f_j)) = \bigvee_j \mathrm{role}(T_j) \quad (4)$$

where, $P_i(T_j)$ is T_j's measurement of P_i, $f_i(T_j)$ is T_j's measurement of f_i, ext(P_i), P_i's measured extension, is the bit-vector of roles for which P_i's has been measured by some T_j to be true, ext$(\triangle(P_i))$ is the bit-vector of roles for which P_i's value has been measured (be it true or false), and role(T_j) is the the role (in bit-vector format) to which T_j is bound. Passive STMs, such as the description buffer, report their data to the inference network in the same way.

3.5. Specifying arguments to behaviors

Behaviors, such as the DRIVETO behavior, take as arguments objects in the world. Ultimately, DRIVETO has to obtain direction and distance information about the object from either a visual tracker bound to the object or an odometric tracker bound to it. It gets this information through its access ports to the visual tracking STM and the odometric tracking STM. It drives the role input of the port with the bit-vector representing X. The access port compares it to the bindings of each tracker in the STM and drives the output of the port with the distance and direction signals of the matching tracker (see figure 2).[3] DRIVETO then uses whatever distance and direction data is presented on the output of the port to steer the robot appropriately.

Our delivery control system included the rule

- If goal(near(X)) $\wedge$ $\triangle$(distance(X)) $\wedge$ $\triangle$(direction(X)), then activate the DRIVETO behavior with the argument X.

which we can implement by computing the compound predicate D, given by

$$D(x) \equiv$$
$$goal(\mathrm{near}(x)) \wedge \triangle\mathrm{distance}(x) \wedge \triangle\mathrm{direction}(x)$$

and routing its extension to the argument input of DRIVETO. Whenever D has a non-null extension and one of the trackers has a binding for one of the objects in its extension, then the tracker will drive the output of DRIVETO's access port with that object's direction and distance information, causing the robot to drive toward it.

An important question is how to resolve conflicts when D's extension has many objects. Thus far, we have dealt with the problem by writing the control rules so it can't happen.

3.6. Equality

Sometimes, it is useful to be able to assert that two roles co-refer. For example, equality assertions could be used to implement an alternative form of parameter passing. Rather than DRIVETO taking a tag as an input, DRIVETO could be designed to drive to any object bound to the role TARGET. DRIVETO could then be "called" on some other object, such as PATIENT, by asserting the equality TARGET=PATIENT.

In the general case, there are n^2 possible assertions of equality between n roles. The current set of equality assertions can be represented as an $n \times n$ matrix $\mathbf{E}$ in which E_{ij} is true iff i and j are asserted to be equal. Given the matrix, we can map a set of names $\mathbf{n}$ to the set of names, $\mathbf{n}_E$, equivalent to it, by performing a matrix multiplication:

$$\mathbf{n}_E = (\mathbf{I} + \mathbf{E})\mathbf{n}$$

Thus we can assert that two names n_i and n_j co-refer by asserting the entries E_{ij} and E_{ji} in the matrix.

In the general case, we must first compute the transitive closure of the asserted equalities. Transitive closure can be computed iteratively by repeated matrix multiplication. Alternatively, if we can place an ordering on the names such that equality assertions are only made between names adjacent in the ordering, then $\mathbf{E}$ will be in block diagonal form and we can compute the mapping directly as:

$$\mathbf{n}_E = (\mathbf{I} + \mathbf{E_1})(\mathbf{I} + \mathbf{E_2})...(\mathbf{I} + \mathbf{E_m})\mathbf{n}$$

where the $\mathbf{E}_1$ are the blocks of $\mathbf{E}$.

One of the advantages of this scheme is that the matrix multiplications need only be performed in the STMs. Each tracker (or other representation) multiplies the set of roles to which it is bound by

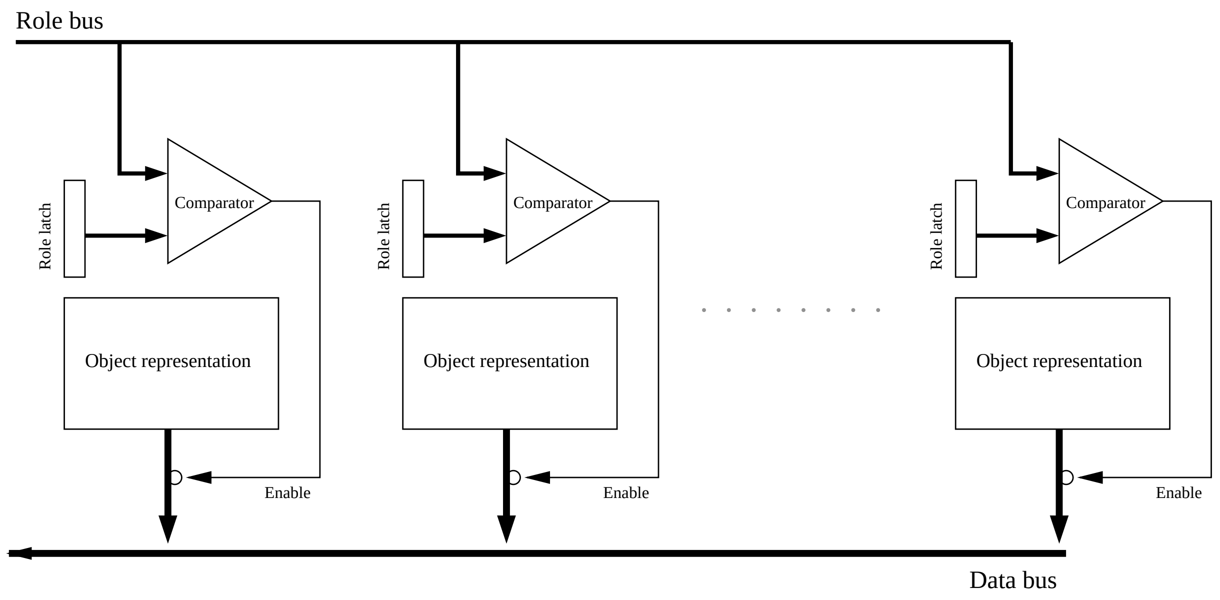

Fig. 2. Access ports. Each object representation (tracker, description, etc.) stores the set of roles to which it is bound in a role latch. When a behavior is activated, its argument is specified by driving the role bus with the argument's role. The tracker whose role latch matches the role bus then drives the output bus with its data.

$\mathbf{I}+\mathbf{E}$ to obtain the set of roles equivalent to the ones to which its bound, then uses those roles in driving the predicate and function busses. Thus, we modify eqs. 1 as follows:

$$\begin{aligned}
\text{ext}(P_i) &= \bigvee_{\{j|P(T_j)\}} (\mathbf{I}+\mathbf{E})\text{role}(T_j) \\
\text{ext}(\triangle(P_i)) &= \bigvee_j (\mathbf{I}+\mathbf{E})\text{role}(T_j) \\
\text{f}_\text{i}(r) &= \begin{cases} f(T_j), & r \in (\mathbf{I}+\mathbf{E})\text{role}(T_j) \\ \text{undefined}, & r \text{ not bound} \end{cases} \\
\text{ext}(\triangle(f_j)) &= \bigvee_j (\mathbf{I}+\mathbf{E})\text{role}(T_j)
\end{aligned}$$

The derived predicates and functions will then be closed under the current set of equality assertions. This lets us avoid computing equality separately at each predicate or function. For t trackers and n roles, the scheme then requires $O(tn^2)$ hardware, or, provided the roles fit in a single machine word, $O(tn)$ time on a von Neumann machine.

4. Implementation

We have implemented role-passing on a vision-based mobile robot called Kludge (see figure 1). Kludge is built from a shortened RWI B14 enclosure with an MIT/DIdeas Cheap Vision Machine serving as its main computer (see figure 1). The CVM is a 25MIP floating-point DSP with 1MB of RAM and a memory-mapped frame grabber. The CVM can perform a wide range of real-time vision operations and consumes only 5-10W. Kludge is also equipped with a pair of servo-controlled mandibles suitable for grasping a set of balls. Its visual and motor capabilities allow it to find and grab balls, bring them to people, or follow people, given knowledge of the color of their clothing. It is also used to play games with children.

Kludge implements the visual tracking STM, the odometric tracking STM, and the description buffer discussed above. Its control rules implement a range of fetching and following tasks, which we describe below. The rules are expressed in a language called *microdoer*, which is similar in spirit to MICROPLANNER or Prolog. The microdoer interpreter simulates the simulated feed-forward network inference network described above and calls the appropriate routines for updating behaviors and perceptual routines. On every cycle of its main control loop, it acquires a new image, runs the vision system and other sensor systems, updates the sensory outputs, reevaluates all the nodes in the inference network, and, finally, runs the behaviors.

Table 1. Roles in the delivery task.

Role	Meaning
Patient	The ball to be delivered
Source	Starting location of the robot
Destination	Person or place to which to deliver the object

```
(define-STM-boundp-node have-description description-STM #f)
(define-STM-boundp-node see tracker-STM have-description)
(define-STM-output-node distance tracker-distance tracker-STM see)
(define-STM-output-node image-x-coordinate tracker-x tracker-STM see)

(define-threshold-node visually-near < distance 13)
(define-threshold-node inhand < distance 3)
(define-test-node visually-facing image-x-coordinate centered?)

(define-STM-predicate odometrically-facing
  odometer-STM odometrically-facing? #f)
(define-STM-predicate odometrically-near
  odometer-STM odometrically-near? #f)
(define-STM-boundp-node odometrically-tracking odometer-STM #f)
```

Fig. 3. Sensory predicates and functions in Kludge.

The microdoer interpreter itself runs under a simple Scheme interpreter written in C. Kludge's vision code is written in C and is called from Scheme. The vision system allows it to search for and track several objects simultaneously. Search and tracking are based on color. The vision system can also perform real-time collision avoidance using the Polly algorithm [19]. Since the Microdoer interpreter is running on a low performance interpreted lisp, it is much slower than it ought to be. The set of rules given here take approximately 50ms to update on each clock cycle. Since the vision system and the generation of debugging output also each take around 50ms, the frame rate of the system varies between 7Hz and 10Hz depending on the amount of debugging output being generated.

The various STMs compute 13 predicates and functions (see figure 3). Most are either *boundp* predicates (true iff a given STM binds a given role), *outputs* (real-valued functions), or *thresholds* (true if a given function is above or below a given threshold).

4.1. Rules for fetch-and-follow tasks

Given these sensory predicates, it is straightforward to write rules to implement fetching, following, and delivering various objects. One rule tells the system that it should use the `driveto` behavior to establish the `near` predicate. Another rule tells it that to achieve the `inhand` predicate, it needs to be near the object and facing it, then open the mandibles and drive forward. A separate set of mandible control rules grab any object that is seen to be in the grabbing area.

Some of the control rules are implemented as conjunctions and disjunctions of sensory predicates (see figure 4). These are computed as in section 3.3. Conjunctions come in two forms depending on whether they spawn their subgoals serially or in parallel. Parallel conjunctions, when goals, always make their conjuncts goals, whereas serial conjunctions first make their first conjunct a goal, then, when it is achieved, make the second conjunct a goal, and so on. Subgoaling is updated on every clock cycle, so if the first conjunct becomes false, the other conjuncts lose their goal

```
(define-disjunction near (visually-near odometrically-near))
(define-disjunction facing (visually-facing odometrically-facing))
(define-disjunction boundp (see have-description odometrically-tracking))

(define-parallel-conjunction grabable (near facing))
(define-serial-conjunction grab (grabable mandibles-open approach))
```

Fig. 4. Other compound predicates in Kludge.

```
;; Performing grab will make an object be in the hand.
(reduce-unsatisfied inhand grab)

;; Run the driveto behavior to get near an object
(when-unsatisfied near driveto)

;; Run the visual-search! behavior when you need to see an object.
(do-all! (logand (unsatisfied-goal see)
 (know-that have-description))
 visual-search!)

;; Control mandibles
(when-unsatisfied mandibles-open open-mandibles!)
(when (not-exists? (know-that near))
  (fold-mandibles!))
(when (exists? (know-that inhand))
  (close-mandibles!))

;; Query the user if they didn't specify a desired role.
(when-unsatisfied boundp query-user)
```

Fig. 5. Explicit action and reduction rules.

status. Disjunctions do not perform goal regression at all.

In addition to conjunctions and disjunctions, there are a set of explicit goal reduction rules linking goals to behaviors or subgoals (see figure 5). The semantics of the rules are as follows:

- (when-unsatisfied P B): when $P(r)$ for some role r is an unsatisfied goal, fire the behavior $B(r)$.
- (reduce-unsatisfied P_1 P_2): when $P_1(r)$ is an unsatisfied goal, assert $goal(P_2(r))$.
- (do-all! P *proc*): run *proc* on every role for which P is true.
- when: as in Common Lisp.

These control rules allow the system to perform a range of actions from simply driving up to a place or object to delivering a specified object to a specified place. The choice of the task is determined by the initial bindings of representations to roles and by the goals asserted in the system. To grab a red object, bind the red description to *patient* and assert $goal(inhand(patient))$. To drive to it without grabbing it, bind it to *destination* and assert $goal(near(destination))$ instead. To drive to a specified place, bind its odometric position to *destination* instead of the red description.

Table 2. Kludge's lexicon

Item	Category	Roles	Action
bring	verb	*patient destination*	assert $goal(inhand(patient))$, then assert $goal(near(destination))$
get	verb	*patient*	assert $goal(inhand(patient))$
go	verb		assert $goal(near(destination))$
come	verb	*destination*	assert $goal(near(destination))$
follow	verb	*patient*	assert $goal(near(patient))$
face	verb	*patient*	assert $goal(facing(patient))$
wait	verb		wait until time in time-buffer
turn	verb		turn number of degrees in turn-buffer
then	conjunction		halt parser until current goal is satisfied
home	noun		bind current location in odometry STM to current role
kludge	noun		bind kludge description to current role
blue	noun		bind blue description to current role
green	noun		bind green description to current role
red	noun		bind red description to current role
pink	noun		bind pink description to current role
black	noun		bind black description to current role
me	noun		bind black description to current role
to	preposition	*destination*	set current role to *destination*
from	preposition	*source*	set current role to *source*
continually	modifier		set don't-terminate flag
seconds	modifier		set time buffer to current time plus number buffer
degrees	modifier		set turn buffer to number buffer
left	modifier		negate turn buffer
right	modifier		none

Table 3. Example execution trace for the command ''bring red to me.'' Red is interpreted as any red object, and me is interpreted as any sufficiently large black object (the human is assumed to be wearing black jeans).

Event or state	Response
$goal(inhand(patient))$	Assert $goal(see(patient))$
$have-description(patient)$	Fire visual-search!$(patient)$
$see(patient)$	Fire $driveto(patient)$
$near(patient) \wedge facing(patient)$	Fire open-mandibles and approach
Patient stolen by human	Close mandibles, abort approach, re-fire visual-search
$see(patient)$	Fire $driveto(patient)$
$near(patient) \wedge facing(patient)$	Fire open-mandibles and approach
$inhand(patient)$	Fire close-mandibles, assert $goal(near(destination))$
$goal(near(destination))$	Assert $goal(K(distance(destination)))$
$goal(K(distance(destination)))$	Assert $goal(see(destination))$
$have-description(destination)$	Fire visual-search!$(destination)$
$see(destination)$	Fire $driveto(destination)$
Dest goes out of view	Abort driveto, fire visual-search!$(destination)$
$see(destination)$	Fire $driveto(destination)$
$near(destination) \wedge facing(destination)$	Fire open-mandibles

4.2. Following instructions

To test out the system, we wrote a simple finite-state parser. While simple, it has been useful for debugging and experimentation.

The parser keeps the user's input in a shift-register and parses one word per control loop cycle. A separate shift register holds the role names that successive syntactic components of the current verb are to take. Encountering a verb asserts a goal and loads the role register. Encountering a noun binds the representation listed in its lexical entry to the current role. Encountering a preposition changes the current role. The conjunction "then" (as in "do A then do B") halts the parser until the current goal is achieved. There are also a few modifiers, such as "continually" which changes verbs from specifying goals of achievement to maintenance goals. As a final feature, the action rule `(when-unsatisfied boundp` query-user) (see figure 5) fires whenever one of the roles of the current verb has been left unspecified. It halts the robot and displays the message "what?" or "where?", depending on the role that was left unbound. If the user replies (i.e. with "home" or "to me"), the parser restarts and fills in the missing binding.

It must be stressed that this is an extremely simplistic model of parsing. It does not support determiners and it treats color words as nouns. It is limited to sentences like:

```
follow me
follow blue
get blue
bring blue here
bring blue to me
bring blue to red
go to blue then wait 10 seconds then
   bring blue to me
```

The system is not meant as a model of human comprehension.

4.3. Example

Table 3 gives an example trace of the system's behavior for the command "bring red to me". Note that unexpected contingencies, such as the human stealing the ball from it or its target straying out of view, do not cause execution failures — they simply cause it to rerun earlier actions (c.f. Nilsson [32]).

5. Discussion

Role-passing is an alternative hybrid control architecture. Unlike previous attempts to extend behavior-based systems [34, 27], it allows limited predicate/argument structure in reasoning and passing of parameters to behaviors. This improves modularity and reduces computational complexity. It allows a single control system to implement "go to the red ball," "go to the blue ball," and "go to odometric location (x,y)" without having separate behaviors for each possible destination. It also makes it practical to write at least a simplistic NL parser.

Unlike tiered architectures [18] which rely on conventional symbolic databases, role-passing has a tractable epistemic model. Behaviors, perceptual operations, and the reasoning system all communicate using a standard mechanism in which objects are referred to by their role tags rather than through lisp symbols (BEE0001) or S-expressions encoding definite descriptions [14]. The bit-vectors used in role passing are easy for the perceptual system to generate and for the behaviors to decode. This makes it practical to completely update the world model on every cycle of the control loop. Since the inference rules can be compiled into a dependency network, the robot can also update all its inferences on every cycle, allowing the system to detect and respond to execution errors or other contingencies immediately.

By representing the agent's goals and state of knowledge explicitly using modals, the system can automatically allocate perceptual attention by regressing knowledge goals through logical connectives to determine the epistemic actions needed to achieve them. This simplifies the design of the system and relieves the programmer of the need to salt her axiomatization with separate epistemic rules stating when and how to check preconditions and postconditions.

The cost of these benefits is reduced expressiveness. Role-passing cannot handle term expressions, binary predicates or functions with arbitrary image sets. It is too weak to express the situation calculus. While some limitations must be accepted in order to assure real-time performance, there may be better trade-offs that can be made.

Certain features, however, appear very difficult to support in any such system. Term expressions require the dynamic allocation and mutation of arbitrary tree-structures, which are extremely difficult to parallelize. This amounts to solving the connectionist *binding problem* [36]. So far as is known, any general solution to implementing arbitrary dynamic tree and pointer structures will either require a large (potentially $O(n^2)$) switching network to route signals, or will impose serial bottlenecks (effectively reducing it to a von Neumann machine). The fact that matching term expressions (unification) is P-complete [10] and is therefore believed to be unparallelizable is not encouraging.[4]

While role passing doesn't support term expressions, it does allow the binding of a small, fixed set of top-level variables, an approach which is gaining popularity. Minsky has proposed implementing communication within systems using global busses with task-specific semantics ("c-lines" [30], and later "pronomes" [31]). Shastri and Ajjanagadde describe a solution to the binding problem that supports a small number of general variables [35]. (Indeed, Shastri uses bit-vector representations for implementing his scheme on von Neumann architectures.) Agre and Chapman [2, 1] discuss the use of indexicals bound in the perceptual system as a replacement for variables bound by switching networks in the central system.

Like Agre and Chapman's deictic representation, role passing requires that the indexical names have functional significance; that they are effectively roles (slots) of what would otherwise be represented in some kind of frame structure. However, Agre and Chapman use extremely specific roles names. Microdoer requires that there be a relatively small number of role names that get recycled from situation to situation. Whereas Agre and Chapman's Pengi system has indexicals like `the-bee-that-is-chasing-me`, a role passing system would have to use a more generic role name like *threat*.

One advantage of giving indexicals specific functional significance (rather than making them more like normal variables) is that it gives us some leverage in representing higher-order predicates. Microdoer cannot represent a relation like $deliver(A, B)$ because its extension would require a quadratic number of bits. By analyzing deliver as

$$deliver \wedge patient = A \wedge destination = B$$

rather than as a two-argument relation, we can represent the concept of delivery. There is still a cost, however: we cannot represent two different instances of delivery at once because they would require inconsistent bindings of *patient* and *destination*. Thus, the technique of modeling binary predicates with indexicals is much less expressive than full first-order logic. It cannot, for example, express the situation calculus. However, it buys a considerable complexity improvement over languages that can (e.g. [25]) and also provides us with a simple epistemology. If the higher arity predicates that arise in practice can't be handled by some simple mechanism like indexicals, then it is unclear how we can ground them efficiently in real time at all.

As mobile robots and active vision systems become more capable, the need for powerful, expressive control languages will grow. A promising approach is to look for maximally expressive languages that are efficiently mappable into dependency networks. Parallel networks give good performance, but, more importantly, can be grounded efficiently in sensors. Role-passing is one example of how this can be done, and microdoer is one example of how a programming language can be built around role-passing. Better architectures and languages will no doubt be be developed as the robotics community attempts increasingly complex tasks.

Acknowledgements

Many thanks to Leslie Kaelbling, Lars Bergstrom, and Ivan Yen, for providing useful comments on this paper. Support for this research was provided in part by the National Science Foundation under grant number IRI–9625041. The Institute for the Learning Sciences was established in 1989 with the support of Anderson Consulting, part of the Arthur Anderson Worldwide Organization.

Notes

1. Real robots almost never use a purely reactive or deliberative system in the sense used here. For example, the subsumption architecture [7] is not a reactive system. In fact, it has been used to build planners [29, 27]. However, they are still useful idealizations and so the debate within the community has been largely cast in terms of them.
2. In reality, the complexity of planning depends on many factors, see [11] for a survey of results on complexity bounds for different forms of planning.
3. This is how it would work in a real hardware implementation. In our serial implementation, behaviors are just subroutines that are called several times a second. Ports are implemented with a subroutine, called by the behavior, that scans the bindings of an STM and returns the output data for the tracker bound to the specified role.

4. Care must be taken in this argument, however. Any problem can be parallelized for bounded size inputs by using an exponential amount of hardware. Technically, the P-completeness result suggests that unification cannot be computed by small, shallow, feed-forward networks.

References

1. Philip E. Agre. The dynamic structure of everyday life. Technical Report 1085, Massachusetts Institute of Technology, Artificial Intelligence Lab, October 1988.
2. Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, 1987.
3. Ronald Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1997.
4. Andrew Blake and Alan Yuille, editors. *Active Vision*. MIT Press, Cambridge, MA, 1992.
5. R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent reactive agents. In JSI [18].
6. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automoation*, 2(1):14–23, March 1986.
7. Rodney A. Brooks and Jonathan H. Connell. Asynchronous distributed control system for a mobile robot. In *Cambridge Symposium on Optical and Optoelectronic Engineering*. SPIE, October 1986.
8. Christopher Brown, David Coombs, and John Soong. Real-time smooth pursuit tracking. In Blake and Yuille [4], pages 126–136.
9. Jill D. Crisman. Color region tracking for vehicle guidance. In Blake and Yuille [4], chapter 7.
10. C. Dwork, P. Kanellakis, and J. C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1(1):35–50, 1984.
11. K. Erol, D.S. Nau, and V.S. Subrahmanian. Complexity, decidability, and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1–2):75–88, July 1995.
12. Scott E. Fahlman. *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, MA, 1979.
13. Stuart M. Fairley, Ian D. Reid, and David W. Murray. Transfer of fixation for an active stereo platform vis affine structure recovery. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 1100–1105, June 1995.
14. R. James Firby. Adaptive execution in complex dynamic worlds. YALEU/CSD/RR 672, Computer Science Department, Yale University, 1989.
15. Gregory D. Hager. Calibration-free visual control using projective invariance. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 1009–1015, June 1995.
16. Takaaki Hasegawa, Yukiko I. Nakano, and Tsuneaki Kato. A collaborative dialog model based on interaction between reactivity and deliberation. In W. Lewis Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, pages 83–87, Marina del Rey, CA USA, February 1997. ACM SIGART, ACM Press.
17. Henry Hexmoor, Ian Horswill, and David Kortenkamp. Software architectures for physical agents. In JSI [18].
18. Henry Hexmoor, Ian Horswill, and David Kortenkamp, editors. *Special issue on software architectures for physical agents, Journal of Theoretical and Experimental AI*. Cambridge University Press, 1997.
19. Ian Horswill. Collision avoidance by segmentation. In *Proceedings of the 1994 IEEE/RSJ Internation Conference on Intelligent Robots and Systems*, Munich, Germany, September 1994. IEEE Press.
20. Ian Horswill. Visual routines and visual search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, August 1995.
21. Daniel P. Huttenlocher, Jae J. Noh, and William J. Rucklidge. Tracking non-rigid objects in complex scenes. TR 93-1320, Computer Science Department, Cornell University, December 1992.
22. Hirochika Inoue. Vision based robot behavior: Tools and testbeds for real world ai research. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 767–773, Chambery, France, August 1993.
23. Leslie P. Kaelbling. Goals as parallel program spcifications. In *Proceedings, AAAI-88*, pages 60–65, St. Paul, MN, 1988.
24. Hector J. Levesque and Ronald J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised edition). In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, pages 42–70. Morgan Kaufman, Los Altos, CA, 1985.
25. Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scher. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 1996.
26. David Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, 8(2):113–122, August 1992.
27. Pattie Maes. How to do the right thing. AI Memo 1180, MIT Artificial Intelligence Laboratory, December 1989.
28. Maja Mataric̀. Behavior-based control: Examples from navigation, learning, and group behavior. In JSI [18].
29. Maja J. Mataric. Minimizing complexity in controlling a collection of mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 830–835, Nice, France, May 1992.
30. Marvin Minsky. Plain talk on neurodevelopmental epistemology. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1083–1092, Cambridge, MA, August 1977.
31. Marvin Minsky. *The Society of Mind*. Simon and Schuster, New York, NY, 1986.
32. Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1994.

33. Peter N. Prokopowicz, Michael J. Swain, and Roger R. Kahn. Task and environment-sensitive tracking. In W. Martin, editor, *Proceedings of the IAPR/IEEE Workshop on Visual Behaviors*, pages 73–78, Seattle, June 1994.
34. Stanley J. Rosenschein and Leslie Pack Kaelbling. The synthesis of machines with provable epistemic properties. In Joseph Halpern, editor, *Proc. Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98. Morgan Kaufmann, 1986.
35. Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16, 1993.
36. Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1–2):159–216, 1990.
37. Jeffrey D. Ullman and Jennifer Wisdom. *A First Course in Database Systems*. Prentice-Hall, 1997.

Ian Horswill is an Assistant Professor of Computer Science at Northwestern University. He received his Ph.D. in Computer Science from the Massachusetts Institute of Technology. His research interests include planning, problem solving, vision, and robotics. However, his principle research focus is on integrated systems that cleanly link perception, reasoning, and action.