Domain-General Simulation and Planning With Physical Schemas

Keywords: planning, simulation, game playing
Tracking number: A804

Abstract

Physical schemas are representations of simple physically grounded relationships and interactions such as "move," "push," and "contain." We believe they are the conceptual primitives an agent employs to understand its environment. Physical schemas can be used at varying levels of abstraction across a variety of domains. We have designed a domain-general agent simulation and control testbed based on physical schemas. If a domain can be described in physical terms as agents moving and applying force, it can be simulated in this testbed. Furthermore, we show that physical schemas can be viewed as abstract plans and can be used as the basis of a domain-general planner. Our simulation and planning system is currently being evaluated in a continuous, dynamic, and adversarial domain based on the game of Capture the Flag.

Physical Schemas

One of the big open questions in cognitive psychology is how humans come to conceptualize the world around them. How do we learn, for example, that a stuffed animal in the shape of a cat will typically remain where it was placed, but that a real cat won't? How do we know that liquids can be transported using cups, buckets, or bowls, but not paper bags? How do we know that any of these objects could be used to transport sand, even though we may never have had experience moving sand with a cup?

In a series of papers, Jean Mandler put forward the notion that infants acquire *image schemas*, preconceptual redescriptions of their sensory input [Mandler, 1988; 1992]. Image schemas are not much more than pattern detectors or filters, but they form the building blocks from which adult concepts develop. One part of the concept of "cat," for example, is the ANIMATE MOTION schema, which captures the particular way animals move.

Influenced by Eleanor Rosch's research on categorization, Lakoff and Johnson argue that categories are based less on objective features such as color, size, and shape, than on *interactional* properties and relationships, such as "graspable" and "fits-in-my-mouth," which characterize how an agent interacts with its environment [Johnson, 1987; Lakoff and Johnson, 1980; Lakoff, 1984]. At the same time, AI researchers such as Agre [Agre, 1988], Chapman [Chapman, 1991], and Ballard [Ballard, 1989] have argued for *deictic* or agent-centered representations. Lakoff and Johnson make a convincing case that the primitive interactional knowledge acquired by infants becomes more elaborate through abstraction and metaphorical extension as the agent develops [Lakoff and Johnson, 1980]. The notion of "grasping," for example, which is presumably learned very early in life, is abstracted by adults to the mental realm and then describes the process of understanding an idea. Just as grasping with your hands involves getting a good grip on an object, preventing it from slipping away, and being able to feel its form and texture, grasping an idea involves "getting your mind" around an abstract concept and getting a feel for its structure. Lakoff and Johnson would argue that the notion of grasping an idea is given *meaning* by the more primitive notion of grasping an object.

Motivated by this research, we gave ourselves the task of exploiting the generality and ubiquity of schemas. It occurred to us some time ago that many of the simulators we had been writing really were just variations on a theme. Physical processes, military engagements, and games such as billiards are all about agents moving and applying force to one another (see, for example, [Tzu, 1988] and [Karr, 1981]). Even the somewhat abstract realm of diplomacy can viewed in these terms: One government might try to *apply pressure* to another for some purpose, or intend to *contain* a crisis before it spreads.

In order to simulate and reason in these domains, we need a set of primitives. We call them *physical schemas*. Like image schemas, they describe basic relationships and interactions between objects. In order to limit their number and solidify their definition, we require them to all be grounded in physics, specifically in the processes of moving and applying force. Examples are **move**, **push**, **reduce**, **contain**, **block**, or **surround**. If moving an army is conceptually no different than moving a robot, both these processes can be represented with one **move** action in a simulator. We believe that people think and solve problems in terms of physical schemas. An example: A person notices his sink is leaking, and considers what to about it. He realizes that there are cracks in the sink. The way to solve the problem is to plug the cracks. Now confront this person with a military problem: Hostile forces are moving across a mountain range into friendly territory. What should be done about it? If the person understands that military forces can behave like water, and that the cracks are passes in the mountain range, he can prevent the flow by making the passes untraversable.

In order to evaluate the feasibility of domain-general schema-based simulation and planning, we have developed a simulator of physical schemas, the Abstract Force Simulator (AFS). It operates with a set of abstract agents, circular objects[1] called "blobs," which have a small set of physical features, including mass, velocity, friction, radius, attack strength, and so on. A blob is an abstract unit; it could be an army, a soldier, or a political entity. Every blob has a small set of primitive actions it can perform, primarily **move** and **apply-force**. All other schemas are built from these actions. Simply by changing the physics of the simulator, that is, how mass is affected by collisions, what the friction is for a blob moving over a certain type of surface, etc., we can and we did turn AFS from a simulator of billiard balls into one of unit movements in a military domain.

AFS: The Abstract Force Simulator

AFS is a simulator of physical processes. It is tick-based, but the ticks are small enough to accurately model the physical interactions between blobs. Although blobs themselves move continuously in 2D space, for reasons of efficiency, the properties of this space, such as terrain attributes, are represented as a discrete grid of rectangular cells. Such a grid of cells is also used internally to bin spatially proximal blobs, making the time complexity of collision detection and blob sensor modeling no greater than linear in terms of the number of blobs in the simulator. AFS was designed from the outset to be able to simulate large numbers (on the order of hundreds or thousands) of blobs.

The physics of the simulation are presently defined by the following parameters:

- Blob-specific attributes:
  - maximum acceleration and deceleration
  - friction of the blob on different surfaces
  - viscosity and elasticity: do blobs pass through one another or bounce off?
- Global parameters:
  - the effect of terrain on blobs
  - the different types of blobs present in the simulation (such as blobs that need sustenance).
  - the damage model: how blobs affect each others' masses by moving through each other or applying force.
  - sustenance model: do blobs have to resupplied in order to prevent them from losing mass?

AFS is an *abstract* simulator; blobs are abstract entities that may or may not have internal structure. AFS allows us to express a blob's internal structure by composing it from smaller blobs, much like an army is composed of smaller organizational units and ultimately individual soldiers. But we don't have to take the internal structure into account when simulating, since at any level of abstraction, every blob is completely characterized by the physical attributes associated with it. Armies can move and apply force just like individual soldiers do. The physics of armies is different than the physics of soldiers, and the time and space scales are different, but the main idea behind AFS is that we can simulate at the "army" level if we so desire—if we believe it is unnecessary or inefficient to simulate in more detail.

Since AFS is basically just simulating physics, the top-level control loop of the simulator is quite straightforward: On each tick, loop over all blobs in the simulator and update each one based on the forces acting on it. If blobs interact, the physics of the world will specify what form their interaction will take. Then update the blob's low-level sensors, if it has any. Each blob is assumed to have a *state reflector*, a data structure that expresses the current state of the blob's sensory experience. It is the simulator's job to update this data structure.

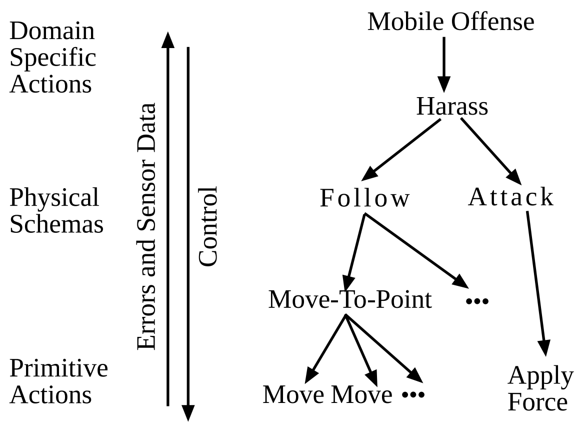

Figure 1: Actions form a hierarchy; control information is passed down, messages and sensor integration occurs bottom-up.

The blob's control architecture is hierarchical. We use the physical primitives **move** and **apply-force** to construct schemas, and schemas to construct domain-specific actions like **convoy** or **sneak-attack** (see Figure 1). Our hierarchy is *supervenient* [Spector and Hendler, 1994]. This means that it abides by the principle that higher levels should provide goals and context for the lower levels, and lower levels provide sensory reports, messages, and errors to the higher levels ("goals down, knowledge up"). A higher level cannot overrule the sensory information provided by a lower level, nor can a lower level interfere with the control of a higher level. Supervenience structures the abstraction process; it allows us to build modular, reusable, actions.

[1]Eventually, blobs will be able to take any shape, and deform and redistribute their mass.

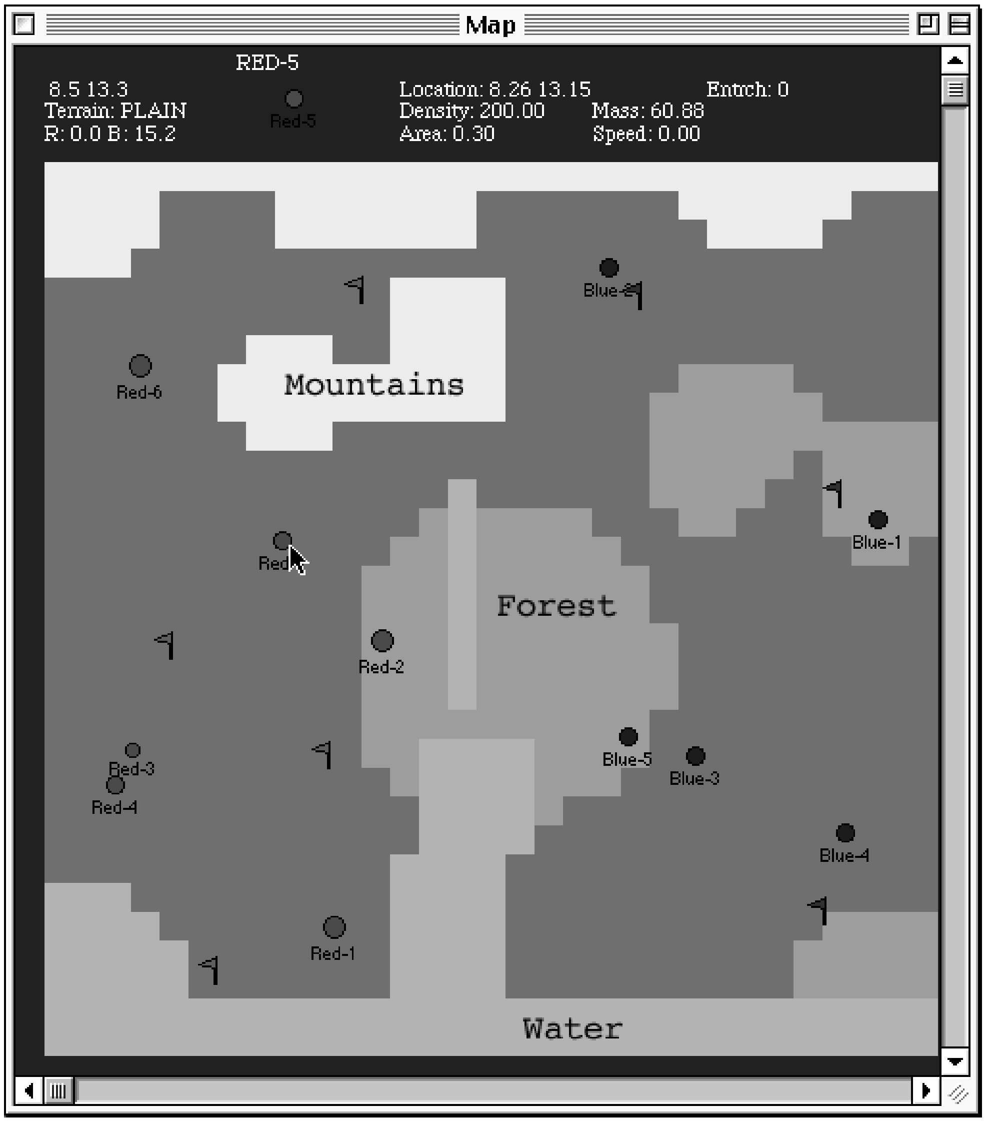

Figure 2: The Capture the Flag domain.

The AFS control architecture provides facilities for sensor management, action scheduling, message passing, and resource arbitration. Since all of AFS's actions are physically grounded, we can even control real-life robots, as long as they adhere to the supervenience principle.

The Capture the Flag Testbed

We have been developing a dynamic and adversarial domain in which to test AFS and the planner. This domain is based on the game of "Capture the Flag" (CTF). In CTF (see Figure 2) there are two teams; each has a number of movable units and flags to protect. Their number and starting locations are randomized. They operate on a map which has different types of terrain. Terrain influences movement speed and forms barriers, which gives us the opportunity to reason about such physical schemas as "blocked" and "constriction." A team wins when it captures all its opponent's flags. A team can also go after its opponent's units to reduce their strength and effectiveness. This game is deceptively simple. The player must allocate forces for attack and defense, and decide which of the opponent's units or flags he should go after. The player must react to plans not unfolding as expected, and possibly retreat or regroup. There are many tactics, from attacking all-out to trying to sneak by the opponent's line of defense. In our current implementation, both players have a global view of the game; when we add limited visibility many more strategies, such as ambushes or traps, will emerge.

Our goal for this domain is to show how one can reason with physical schemas. Beating the computer is already non-trivial, and as our plans become more refined, it

```
An action or a plan posts a goal G. This invokes the following process:
 1. Search the list of plans for those that can satisfy G.
 2. Evaluate each potential plan's pre-conditions (including dynamics), and only keep those whose pre-conditions
    match.
 3. For each plan, do the following:
    3.1 If the plan requires multiple tasks to be achieved,
        3.1.1 Generate and rank the task list.
        3.1.2 Using heuristics, generate a set of schema lists that achieve these tasks efficiently. Each set of schemas
              is a plan.
    3.2 Evaluate the plan (or set of plans) using forward simulation.
 4. Execute the plan that in simulation results in a world state with the highest score.
```

Figure 3: The planning algorithm.

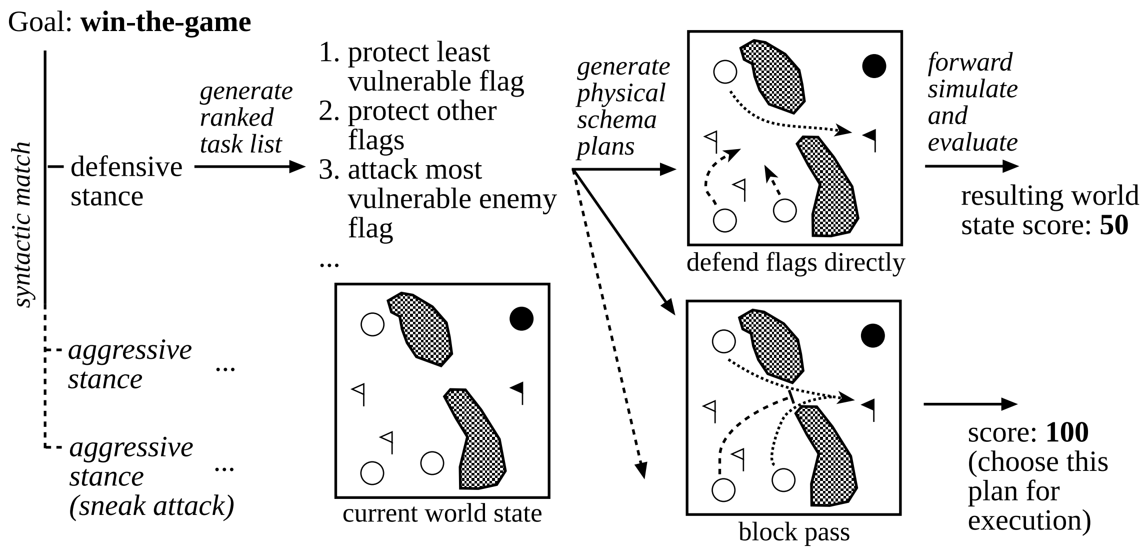
Goal: **win-the-game**
syntactic match
defensive stance
aggressive stance
aggressive stance (sneak attack)
generate ranked task list
1. protect least vulnerable flag
2. protect other flags
3. attack most vulnerable enemy flag
...
current world state
generate physical schema plans
defend flags directly
block pass
forward simulate and evaluate
resulting world state score: **50**
score: **100** (choose this plan for execution)

Figure 4: A planning example: White is trying to satisfy the goal **win-the-game**. Several top-level plans match this goal; the example explores what happens when **defensive-stance** is expanded. In the context of this plan, defense is considered vital, which is reflected in the task list that is generated. There are several sets of schemas that achieve these tasks, and many ways to allocate resources to these schemas. The planner uses heuristics to prune this set. In the first case, two blobs are allocated to flag defense, and one is sent out to attack. In the second plan, only one blob is needed to block the mountain pass, thus protecting the flags, leaving two blobs for the attack. This plan is more likely to succeed and is ranked higher.

will only get harder. Furthermore, the actions we use and a lot of the reasoning we do during planning are not specific to the Capture the Flag domain. In fact, since we already had implementations of **move-to-point** and **attack** from other domains, getting the first Capture the Flag scenario up and running was a fairly painless process.

The Capture the Flag Planner

The CTF planner is a partial hierarchical planner [Georgeff and Lansky, 1986]. By having a set of pre-compiled skeletal solutions, we can avoid the enormous branching factor a generative planner would face in this domain. Partial hierarchical planning meshes very well with the idea that people understand and reason about the world in terms of physical schemas. Viewed at the level of physical schemas, there are only a few different ways to solve a problem. For example, if A and B are point masses, A can cause B to move by i) pushing it, ii) asking it to move (if it an intentional agent), iii) changing the environment so it is forced to move, or iv) initiating movement in B. These separate solutions can be written down as plans that satisfy the goal "make B move." The exciting thing about planning at the physical schema level is that the plans you use are not limited to just one domain. If you can figure out what "move" and "push" *mean* in a domain, you can use your old plans.

A problem in CTF and any other domain that involves the coordination of multiple agents is resource arbitration. Winning CTF involves multiple tasks: protecting your own flags, thwarting enemy offensives, choosing the most vulnerable enemy flag for a counter-attack, and so on. Each requires resources (blobs) to be accomplished. Sometimes one resource can be used to achieve several tasks. For instance, if two flags are close together, one blob might protect both. Or, advancing towards an opponent's flag might also force the opponent to retreat, thus relieving some pressure on one's own flags.

The CTF planner solves the resource allocation problem by first ranking the list of tasks that need to be achieved to satisfy a goal. It then generates several lists of physical schemas that achieve the most important tasks. Examples for schemas are "attack flag C" or "block the mountain pass." Heuristics, such as "prefer schemas that minimize the total number of blobs needed," are used to keep the task list small. Each schema list constitutes one *plan* for achieving the list of tasks. By design, plans cannot contain resource conflicts, since every pertinent task was considered during their generation. If resource problems arise *during* a plan's execution, for example because a blob was destroyed and the schema using it cannot succeed without it, a resource error message is sent to the plan initiator, possibly causing resources to be re-assigned or a complete replan to take place. The complete planning algorithm is outlined in Figure 3. Figure 4 shows an example of the plan generation procedure.

Plan Evaluation Using Critical Points

When several plans apply, partial hierarchical planners typically select one according to heuristic criteria. Military planners will actually play out a plan and how the opponent might react to it. A wargame is a qualitative simulation. The CTF planner does the same: it simulates potential plans at some abstract level, then applies a static evaluation function to select the best plan. The static evaluation function incorporates such factors as relative strength and number of captured and threatened flags of both teams, to describe how desirable this future world state is.

Simulation is a costly operation, and in order to do it efficiently, CTF must be able to jump ahead to times when interesting events take place in the world. The problem that CTF faces is having to impose "states" on a continuous domain. It does this by defining state boundaries, called *critical points*, that are established dynamically, as the plan simulation unfolds. A critical point is a time during the execution of an action or plan where a decision might be made. If this decision can be made at any time during an interval, it is the *latest* such time.

Simple actions, such as moving from point A to point B, only have one critical point: the time at which the action completes. This is the time at which a new decision has to be made about what to do with the blob that was moving. The critical time can easily be estimated given the terrain and the blob's typical movement speed.

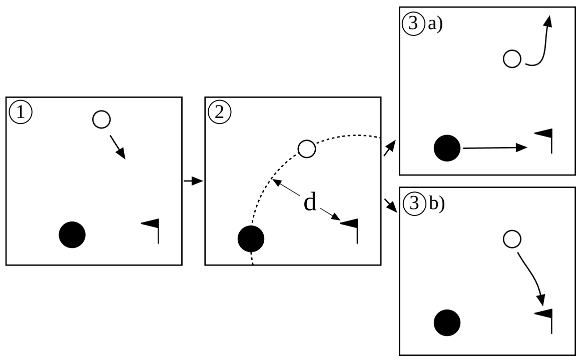

Figure 5: An example for a critical point while executing an attack action.

More complicated actions have larger critical point sets. The attack action depicted in Figure 5 makes a *decision* during its execution: it will abandon the attack if the thing being attacked is protected by a blob larger than the attacker. In this example, a white blob is attacking a black flag and there is a large black blob nearby. The critical point is the time at which the white blob is closer to the flag than the black blob is now. This is the latest point in time at which Black could interfere with the attack action. If Black has started moving to the flag by this time, White will abandon the attack. If Black has remained stationary or gone somewhere else, the attack will be successful and the flag will be destroyed.

Note that critical points are only bounds, they are not the exact times at which a decision will be made. In the above example, the black blob might move to protect its flag right away, in which case White will abandon the attack sooner than the critical time. This is not a large qualitative difference to the scenario that was simulated. If we had simulated without critical points, and simply completed White's action, there *would* have been a large qualitative error: The flag would have either marked as destroyed (which wouldn't have happened if Black moved in), or White would have been destroyed by the protecting black force (which would never have happened since White would have fled before it came to that).

Critical points are essential for plan evaluation in the CTF planner, since they are used to guide forward simulation. The basic idea behind forward simulation is that instead of advancing the world tick by tick, which is time-consuming, we jump right to the next critical point. Forward simulation proceeds in the following way:

1. Add the plan P to be evaluated to all the actions currently ongoing in the simulator.
2. In simulation, loop either until a fixed time in the future or until too many errors have accumulated in the simulation:
   2.1 Compute the minimum critical time t of all actions being simulated.
   2.2 Advance all actions by t time units.

3. Evaluate the resulting world state; return this value as the score for the plan P.

For this algorithm to work, every action and plan must have two functions associated with it. The first computes the next critical time for this action. The second, **(advance** t**)**, takes as an argument a time parameter t and will change the world state to reflect the execution of this action t time units into the future. Currently, we have no automated way of generating these functions, so they are written by the designer of the action. In the case of the attack action shown in Figure 5, the decision about whether or not to abort depends on whether the white blob can get closer to the black flag than any black blob. A simple approximation of the critical point would be the time it will most likely take, given the terrain, to get as close to the black flag as the closest black blob is now. A more accurate approximation would take the current velocities of all black blobs into account, since some might be moving towards the flag.

Forward simulation advances to the minimum t_{min} of all critical points, since the computation of the next critical time for the action with the minimal critical time will depend on the state of the world at time t_{min}. In the extreme case—when many actions are being simulated or when many critical times are underestimated—the evaluation process can degenerate into tick-based simulation.

CTF currently has no explicit opponent model. We simply assume the opponent would do what we would do in his situation. During forward simulation, the action list also contains opponent actions. When CTF starts plan evaluation, it simply puts the top-level goal **win-the-game** for the opponent into the action list. The opponent action's critical times are computed just like ours, and they are advanced in the same way. Whereas our side evaluates all plans and chooses the best one, the opponent chooses the worst one (for us). This really is a form of minimax search, with the two sides executing their plans in parallel.

Summary and Discussion

Physical schemas are domain-general actions and relationships based on the physics of how agents interact. Out of these schemas we have a built a general simulator, AFS. It can be used in any domain that can be described as agents moving and applying force to one another. We have also seen how the physics of a domain naturally prescribes a set of plans, and we have introduced a partial hierarchical planner that takes advantage of these plans. In accordance with our view that physical schemas are able to describe the world at varying levels of abstraction, we simulate at a more abstract level to evaluate plans.

The evaluation of our claims of domain-generality are still in progress. While AFS has been used in a number of domains with different underlying physics, we have yet to show how easily physical schema plans are transferable from one to domain to another. It is our hope, however, that we can establish a general plan hierarchy for any physical domain based on a general causal model of the physics we are simulating.

Acknowledgments

This research is supported by DARPA/USAF under contract numbers N66001-96-C-8504, F30602-97-1-0289, and F30602-95-1-0021. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Defense Advanced Research Projects Agency/Air Force Materiel Command or the U.S. Government.

References

[Agre, 1988] Philip E. Agre. The dynamic structure of everyday life. Technical Report 1085, MIT Artificial Intelligence Laboratory, Cambridge MA, 1988.

[Ballard, 1989] D.H. Ballard. Reference frames for animate vision. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligenc.* Morgan Kaufmann Publishers, Inc., 1989.

[Chapman, 1991] David Chapman. *Vision, Instruction and Action.* MIT Press, 1991.

[Georgeff and Lansky, 1986] Michael P. Georgeff and Amy L. Lansky. Procedural knowledge. *IEEE Special Issue on Knowledge Representation*, 74(10):1383–1398, 1986.

[Johnson, 1987] Mark Johnson. *The Body in the Mind.* University of Chicago Press, 1987.

[Karr, 1981] Alan F. Karr. Lanchester attrition processes and theater-level combat models. Technical report, Institute for Defense Analyses, Program Analysis Division, Arlington, VA., 1981.

[Lakoff and Johnson, 1980] George Lakoff and Mark Johnson. *Metaphors We Live By.* University of Chicago Press, 1980.

[Lakoff, 1984] George Lakoff. *Women, Fire, and Dangerous Things.* University of Chicago Press, 1984.

[Mandler, 1988] Jean M. Mandler. How to build a baby: On the development of an accessible representational system. *Cognitive Development*, 3:113–136, 1988.

[Mandler, 1992] Jean M. Mandler. How to build a baby: II. Conceptual primitives. *Psychological Review*, 99(4):587–604, 1992.

[Spector and Hendler, 1994] Lee Spector and James Hendler. The use of supervenience in dynamic-world planning. In Kristian Hammond, editor, *Proceedings of The Second International Conference on Artificial Intelligence Planning Systems*, pages 158–163, 1994.

[Tzu, 1988] Sun Tzu. *The Art of War.* Shambhala Publications, 1988.