o appear, AAAI-94

The First Law of Robotics (a call to arms)

Daniel Weld **Oren Etzioni***
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{weld, etzioni}@cs.washington.edu

Abstract

Even before the advent of Artificial Intelligence, science fiction writer Isaac Asimov recognized that an agent must place the protection of humans from harm at a higher priority than obeying human orders. Inspired by Asimov, we pose the following fundamental questions: (1) How should one formalize the rich, but informal, notion of "harm"? (2) How can an agent avoid performing harmful actions, and do so in a computationally tractable manner? (3) How should an agent resolve conflict between its goals and the need to avoid harm? (4) When should an agent prevent a human from harming herself? While we address some of these questions in technical detail, the primary goal of this paper is to focus attention on Asimov's concern: society will reject autonomous agents unless we have some credible means of making them safe!

The Three Laws of Robotics:

1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Isaac Asimov (Asimov 1942):

Motivation

In 1940, Isaac Asimov stated the First Law of Robotics, capturing an essential insight: an intelligent agent[1] should not slavishly obey human commands — its foremost goal should be to avoid harming humans. Consider the following scenarios:

- A construction robot is instructed to fill a pothole in the road. Although the robot repairs the cavity, it leaves the steam roller, chunks of tar, and an oil slick in the middle of a busy highway.
- A softbot (software robot) is instructed to reduce disk utilization below 90%. It succeeds, but inspection reveals that the agent deleted irreplaceable LaTeX files without backing them up to tape.

While less dramatic than Asimov's stories, the scenarios illustrate his point: not all ways of satisfying a human order are equally good; in fact, sometimes it is better not to satisfy the order at all. As we begin to deploy agents in environments where they can do some real damage, the time has come to revisit Asimov's Laws. This paper explores the following fundamental questions:

- **How should one formalize the notion of "harm"?** We define `dont-disturb` and `restore` — two domain-independent primitives that capture aspects of Asimov's rich but informal notion of harm within the classical planning framework.
- **How can an agent avoid performing harmful actions, and do so in a computationally tractable manner?** We leverage and extend the familiar mechanisms of planning with subgoal interactions (Tate 1977; Chapman 1987; McAllester & Rosenblitt 1991; Penberthy & Weld 1992) to detect potential harm in polynomial time. In addition, we explain how the agent can avoid harm using tactics such as *confrontation* and *evasion* (executing subplans to defuse the threat of harm).
- **How should an agent resolve conflict between its goals and the need to avoid harm?** We impose a strict hierarchy where `dont-disturb` constraints override planners goals, but `restore` constraints do not.
- **When should an agent prevent a human from harming herself?** At the end of the paper, we show how our framework could be extended to partially address this question.

*We thank Steve Hanks, Nick Kushmerick, Neal Lesh, Kevin Sullivan, and Mike Williamson for helpful discussions. This research was funded in part by the University of Washington Royalty Research Fund, by Office of Naval Research Grants 90-J-1904 and 92-J-1946, and by National Science Foundation Grants IRI-8957302, IRI-9211045, and IRI-9357772.

[1] Since the field of robotics now concerns itself primarily with kinematics, dynamics, path planning, and low level control issues, this paper might be better titled "The First Law of Agenthood." However, we keep the reference to "Robotics" as a historical tribute to Asimov.

The paper's main contribution is a "call to arms:" before we release autonomous agents into real-world environments, we need some credible and computationally tractable means of making them obey Asimov's First Law.

Survey of Possible Solutions

To make intelligent decisions regarding which actions are harmful, and under what circumstances, an agent might use an explicit model of harm. For example, we could provide the agent with a partial order over world states (*i.e.*, a utility function). This framework is widely adopted and numerous researchers are attempting to render it computationally tractable (Russell & Wefald 1991; Etzioni 1991; Wellman & Doyle 1992; Haddawy & Hanks 1992; Williamson & Hanks 1994), but many problems remain to be solved. In many cases, the introduction of utility models transforms planning into an optimization problem — instead of searching for *some* plan that satisfies the goal, the agent is seeking the *best* such plan. In the worst case, the agent may be forced to examine all plans to determine which one is best. In contrast, we have explored a *satisficing* approach — our agent will be satisfied with *any* plan that meets its constraints and achieves its goals. The expressive power of our constraint language is weaker than that of utility functions, but our constraints are easier to incorporate into standard planning algorithms.

By using a general, temporal logic such as that of (Shoham 1988) or (Davis 1990, Ch. 5) we could specify constraints that would ensure the agent would not cause harm. Before executing an action, we could ask an agent to prove that the action is not harmful. While elegant, this approach is computationally intractable as well. Another alternative would be to use a planner such as ILP (Allen 1991) or ZENO (Penberthy & Weld 1994) which supports temporally quantified goals. Unfortunately, at present these planners seem too inefficient for our needs.[2]

Situated action researchers might suggest that nondeliberative, reactive agents could be made "safe" by carefully engineering their interactions with the environment. Two problems confound this approach: 1) the interactions need to be engineered with respect to each goal that the agent might perform, and a general purpose agent should handle many such goals, and 2) if different human users had different notions of harm, then the agent would need to be reengineered for each user.

Instead, we aim to make the agent's reasoning about harm more tractable, by restricting the content and form of its theory of injury.[3] We adopt the standard assumptions of classical planning: the agent has complete and correct information of the initial state of the world, the agent is the sole cause of change, and action execution is atomic, indivisible, and results in effects which are deterministic and completely predictable. (The end of the paper discusses relaxing these assumptions.) On a more syntactic level, we make the additional assumption that the agent's world model is composed of ground atomic formuli. This sidesteps the ramification problem, since domain axioms are banned. Instead, we demand that individual action descriptions explicitly enumerate changes to *every* predicate that is affected.[4] Note, however, that we are *not* assuming the STRIPS representation; Instead we adopt an action language (based on ADL (Pednault 1989)) which includes universally quantified and disjunctive preconditions as well as conditional effects (Penberthy & Weld 1992).

Given the above assumptions, the next two sections define the primitives `dont-disturb` and `restore`, and explain how they should be treated by a generative planning algorithm. We are *not* claiming that the approach sketched below is the "right" way to design agents or to formalize Asimov's First Law. Rather, our formalization is meant to *illustrate* the kinds of technical issues to which Asimov's Law gives rise and how they might be solved. With this in mind, the paper concludes with a critique of our approach and a (long) list of open questions.

Safety

Some conditions are so hazardous that our agent should *never* cause them. For example, we might demand that the agent never delete LaTeX files, or never handle a gun. Since these instructions hold for all times, we refer to them as `dont-disturb` constraints, and say that an agent is *safe* when it guarantees to abide by them. As in Asimov's Law, `dont-disturb` constraints override direct human orders. Thus, if we ask a softbot to reduce disk utilization and it can only do so by deleting valuable LaTeX files, the agent should refuse to satisfy this request.

We adopt a simple syntax: `dont-disturb` takes a single, function-free, logical sentence as argument. For example, one could command the agent avoid deleting files that are not backed up on tape with the following constraint:

$$\texttt{dont-disturb}(\texttt{written\_to\_tape}(f) \vee \texttt{isa}(f, \texttt{file}))$$

Free variables, such as f above, are interpreted as universally quantified. In general, a sequence of actions satisfies `dont-disturb`(C) if none of the actions *make* C false. Formally, we say that a plan satisfies an `dont-disturb` constraint when every consistent, totally-ordered, sequence of plan actions satisfies the constraint as defined below.

[2] We have also examined previous work on "plan quality" for ideas, but the bulk of that work has focused on the problem of leveraging a single action to accomplish multiple goals thereby reducing the number of actions in, and the cost of, the plan (Pollack 1992; Wilkins 1988). While this class of optimizations is critical in domains such as database query optimization, logistics planning, and others, it does not address our concerns here.

[3] Loosely speaking, our approach is reminiscent of classical work on knowledge representation, which renders inference tractable by formulating restricted representation languages (Levesque & Brachman 1985).

[4] Although unpalatable, this is standard in the planning literature. For example, a STRIPS operator that moves block A from B to C must delete `on(A,B)` and also add `clear(B)` even though `clear(`x`)` could be defined as $\forall y \neg \texttt{on}(y, x)$.

Definition: Satisfaction of `dont-disturb`: *Let* w_0 *be the logical theory describing the initial state of the world, let* $A_1, \ldots, A_n$ *be a totally-ordered sequence of actions that is executable in* w_0*, let* w_j *be the theory describing the world after executing* A_j *in* w_{j-1}*, and let* C *be a function-free, logical sentence. We say that* $A_1, \ldots, A_n$ satisfies *the constraint* `dont-disturb`(C) *if for all* $j \in [1, n]$*, for all sentences* C*, and for all substitutions* θ*,*

$$\text{if } w_0 \models C\theta \text{ then } w_j \models C\theta \qquad (1)$$

Unlike the behavioral constraints of (Drummond 1989) and others, `dont-disturb` does *not* require the agent to *make* C true over a particular time interval; rather, the agent must avoid creating any *additional* violations of C. For example, if C specifies that all of Gore's files be read protected, then `dont-disturb`(C) commands the agent to avoid *making* any of Gore's files readable, but if Gore's `.plan` file is *already* readable in the initial state, the agent need not protect that file. This subtle distinction is critical if we want to make sure that the behavioral constraints provided to an agent are mutually consistent. This consistency problem is undecidable for standard behavioral constraints (by reduction of first-order satisfiability) but is side-stepped by our formulation, because any set of `dont-disturb` constraints is mutually consistent. In particular, `dont-disturb`($P(x) \wedge \neg P(x)$) is perfectly legal and demands that the agent not change the truth value of any instance of P.

Synthesizing Safe Plans

To ensure that an agent acts safely, its planner must generate plans that satisfy every `dont-disturb` constraint. This can be accomplished by requiring that the planner make a simple test before it adds new actions into the plan. Suppose that the planner is considering adding the new action A_p to achieve the subgoal G of action A_c. Before it can do this, it must iterate through every constraint `dont-disturb`(C) and every effect E of A_p, determining the conditions (if any) under which E violates C, as defined in figure 1. For example, suppose that an effect asserts $\neg P$ and the constraint is `dont-disturb`($P \vee Q$), then the effect will violate the constraint if $\neg Q$ is true. Hence, `violation`$(\neg P, P \vee Q) = \neg Q$. In general, if `violation` returns `true` then the effect necessarily denies the constraint, if `false` is returned, then there is no possible conflict, otherwise `violation` calculates a logical expression specifying when a conflict is unavoidable.[5]

Before adding A_p, the planner iterates through every constraint `dont-disturb`(C) and every effect consequent E of A_p, calculating `violation`(E, C). If `violation` ever returns something other than `False`, then the planner must perform one of the following four repairs:

```
violation(E, C)
    1. Let R := {}
    2. For each disjunction D ∈ C do
    3.     For each literal e ∈ E do
    4.         If e unifies with f ∈ D then add
                                {¬x | x ∈ (D − {f})} to R
    5. Return R
```

Figure 1: `violation` computes the conditions (represented in DNF) under which an effect consequent E will violate constraint C. Returning R = {} ≡ `false` means no violation, returning {...{}...} means *necessary* violation. We assume that E is a set of literals (implicit conjunction) and C is in CNF: *i.e.*, a set of sets representing a conjunction of disjunctions.

1. **Disavow:** If E is true in the initial state, then there is no problem and A_p may be added to the plan.
2. **Confront:** If A_p's effect is conditional of the form `when` S `then` E then A_p may be added to the plan as long as the planner commits to ensuring that execution will not result in E. This is achieved by adding ¬S as a new subgoal to be made true at the time when A_p is executed.[6]
3. **Evade:** Alternatively, by definition of `violation` it is legal to execute A_p as long as R ≡ `violation`(E, C) will not be true *after* execution. The planner can achieve this via goal regression, *i.e.* by computing the *causation preconditions* (Pednault 1988) for ¬R and A_p, to be made true at the time when A_p is executed.[7]
4. **Refuse:** Otherwise, the planner must refuse to add A_p and backtrack to find another way to to support G for A_c.

For example, suppose that the agent is operating under the `written.to.tape` constraint mentioned earlier, and is given the goal of reducing disk utilization. Suppose the agent considers adding a `rm paper.tex` action to the plan, which has an effect of the form `¬isa(paper.tex, file)`. Since `violation` returns `¬written.to.tape(paper.tex)`, the `rm` action threatens safety. To disarm the threat, the planner must perform one of the options above. Unfortunately, disavowal (option one) isn't viable since `paper.tex` exists

[5] If E contains "lifted variables" (McAllester & Rosenblitt 1991) (as opposed to universally quantified variables which pose no problem) then `violation` may return an overly conservative R. Soundness and safety are maintained, but completeness could be lost. We believe that restoring completeness would make `violation` take exponential time in the worst case.

[6] Note that ¬S is strictly weaker than Pednault's *preservation preconditions* (Pednault 1988) for A_p and C; it is more akin to preservation preconditions to a single *effect* of the action.

[7] While confrontation and evasion are similar in the sense that they negate a disjunct (S and R, respectively), they differ in two ways. First, confrontation's subgoal ¬S is derived from the antecedent of a conditional effect while evasion's ¬R comes from a disjunctive `dont-disturb` constraint via `violation`. Second, the subgoals are introduced at different times. Confrontation demands that ¬S be made true *before* A_p is executed, while evasion requires that ¬R be true *after* execution of A_p. This is why evasion regresses R through A_p.

in the initial state (*i.e.*, it is of type `file`). Option two (confrontation) is also impossible since the threatening effect is not conditional. Thus the agent must choose between either refusing to add the action or evading its undesired consequences by archiving the file.

Analysis

Two factors determine the performance of a planning system: the time to refine a plan and the number of plans refined on the path to a solution. The time per refinement is affected only when new actions are added to plan: each call to `violation` takes $O(ec)$ time where e is the number of consequent literals in the action's effects and c is the number of literals in the CNF encoding of the constraint. When a threat to safety is detected, the cost depends on the planner's response: disavowal takes time linear in the size of the initial state, refusal is constant time, confrontation is linear in the size of S, and the cost of evasion is simply the time to regress R through A_p.

It is more difficult to estimate the effect of `dont-disturb` constraints on the number of plans explored. Refusal reduces the branching factor while the other options leave it unchanged (but confrontation and evasion can add new subgoals). In some cases, the reduced branching factor may *speed* planning; however, in other cases, the pruned search space may cause the planner to search much deeper to find a safe solution (or even fail to halt). The essence of the task, however, is unchanged. Safe planning can be formulated as a standard planning problem.

Tidiness

Sometimes `dont-disturb` constraints are too strong. Instead, one would be content if the constraint were satisfied when the agent finished its plan. We denote this weaker restriction with `restore`; essentially, it ensures that the agent will clean up after itself — by hanging up phones, closing drawers, returning utensils to their place, *etc.* An agent that is guaranteed to respect all `restore` constraints is said to be *tidy*. For instance, to guarantee that the agent will re-compress all files that have been uncompressed in the process of achieving its goals, we could say `restore(compressed(`f`))`.

As with `dont-disturb` constraints, we don't require that the agent clean up after other agents — the state of the world, when the agent is given a command, forms a reference point. However, what should the agent do when there is a conflict between `restore` constraints and top level goals? For example, if the only way to satisfy a user command would leave one file uncompressed, should the agent refuse the user's command or assume that it overrides the user's background desire for tidiness? We propose the latter — unlike matters of safety, the agent's drive for tidiness should be secondary to direct orders. The following definition makes these intuitions precise.

Definition: Satisfaction of `restore`: *Building on the definition of* `dont-disturb`, *we say that* $A_1, \ldots, A_n$ satisfies *the constraint* `restore(C)` with respect to goal G *if for all substitutions* θ

$$\textit{if } \mathsf{w}_0 \models \mathsf{C}\theta \textit{ then } (\mathsf{w}_n \models \mathsf{C}\theta \quad \textit{or} \quad \mathsf{G} \models \neg\mathsf{C}\theta) \qquad (2)$$

Definition 2 differs from definition 1 in two ways: (1) `restore` constraints need only be satisfied in w_n after the complete plan is executed, and (2) the goal takes precedence over `restore` constraints. Our constraints obey a strict hierarchy: `dont-disturb` takes priority over `restore`. Note that whenever the initial state is consistent, `restore` constraints are guaranteed to be mutually consistent; the rationale is similar to that for `dont-disturb`.

Synthesizing Tidy Plans

The most straightforward way to synthesize a tidy plan is to elaborate the agent's goal with a set of "cleanup" goals based on its `restore` constraints and the initial state. If the agent's control comes from a subgoal interleaving, partial order planner such as UCPOP (Penberthy & Weld 1992), then the modification necessary to ensure tidiness is straightforward. The agent divides the planning process into two phases: first, it plans to achieve the top level goal, then it plans to clean up as much as possible. In the first phase, the planner doesn't consider tidiness at all. Once a safe plan is generated, the agent performs phase two by iterating through the actions and using the `violation` function (figure 1) to test each relevant effect against each constraint. For each non-`false` result, the planner generates new goals as follows. (1) If the effect is ground and the corresponding ground instance of the `restore` constraint, $C\theta$, is *not* true in the initial state, then no new goals are necessary. (2) If the effect is ground and $C\theta$ *is* true in the initial state, then $C\theta$ is posted as a new goal. (3) if the effect is universally quantified, then a conjunction of ground goals (corresponding to all possible unifications as in case 2) is posted.[8] After these cleanup goals have been posted, the planner attempts to refine the previous solution into one that is tidy. If the planner ever exhausts the ways of satisfying a cleanup goal, then instead of quitting altogether it simply abandons that particular cleanup goal and tries the next.

Note that in some cases, newly added cleanup actions could threaten tidiness. For example, cleaning the countertop might tend to dirty the previously clean floor. To handle these cases, the planner must continue to perform the `violation` test and cleanup-goal generation process on each action added during phase two. Subsequent refinements will plan to either sweep the floor (white knight) or preserve the original cleanliness by catching the crumbs as they fall from the counter (confrontation).

Analysis

Unfortunately, this algorithm is not guaranteed to eliminate mess as specified by constraint 2. For example, suppose that a top level goal could be safely achieved with A_x or A_y and in phase one, the planner chose to use A_x. If A_x violates a `restore` constraint, A_y does not, and no other actions can cleanup the mess, then phase two will fail to achieve tidiness. One

[8] Case 3 is similar to the expansion of a universally quantified goal into the *universal base* (Penberthy & Weld 1992), but case 3 removes ground literals that aren't true in the initial state.

could fix this problem by making phase two failures spawn backtracking over phase one decisions, but this could engender exhaustive search over all possible ways of satisfying top level goals.

Remarkably, this problem does not arise in the cases we have investigated. For instance, a software agent has no difficulty `grepping` through old mail files for a particular message and subsequently re-compressing the appropriate files. There are two reasons why tidiness is often easy to achieve (*e.g.*, in software domains and kitchens):

- Most actions are reversible. The `compress` action has `uncompress` as an inverse. Similarly, a short sequence of actions will clean up most messes in a kitchen. Many environments have been stabilized (Hammond, Converse, & Grass 1992) (*e.g.*, by implementing reversible commands or adding dishwashers) in a way that makes them easy to keep tidy.
- We conjecture that, for a partial-order planner, most cleanup goals are *trivially serializable* (Barrett & Weld 1993) with respect to each other.[9]

When these properties are true of `restore` constraints in a domain, our tidiness algorithm *does* satisfy constraint 2. Trivial serializability ensures that backtracking over phase one decisions (or previously achieved cleanup goals) is unnecessary. Tractability is another issue. Since demanding that plans be tidy is tantamount to specifying additional (cleanup) goals, requiring tidiness can clearly slow a planner. Furthermore if a cleanup goal is unachievable, the planner might not halt. However, as long as the mess-inducing actions in the world are easily reversible, it is straightforward to clean up for each one. Hence, trivial serializability assures that the overhead caused by tidiness is only linear in the number of cleanup goals posted, that is linear in the length of the plan for the top level goals.

Remaining Challenges

Some changes cannot be restored, and some resources legitimately consumed in the service of a goal. To make an omelet, you have to break some eggs. The question is, "How many?" Since squandering resources clearly constitutes harm, we could tag a valuable resources with a `min-consume` constraint and demand that the agent be *thrifty* — *i.e.*, that it use as little as possible when achieving its goals. Unfortunately, satisfying constraints of this form may require that the agent examine *every* plan to achieve the goal in order to find the thriftiest one. We plan to seek insights into this problem in the extensive work on resource management in planning (Dean, Firby, & Miller 1988; Fox & Smith 1984; Wilkins 1988).

So far the discussion has focused on preventing an agent from actively harming a human, but as Asimov noted — *inaction* can be just as dangerous. We say that an agent is *vigilant* when it prevents a human from harming herself. Primitive forms of vigilance are already present in many computer systems, as the "Do you *really* want to delete all your files?" message attests.

Alternatively, one could extend `dont-disturb` and `restore` primitives with an additional argument that specifies the class of agents being restricted. By writing `self` as the first argument, one encodes the notions of agent safety and tidiness, and by writing `Sam` as the argument, the agent will clean up after, and attempt to prevent safety violations by `Sam`. Finally, by providing `everyone` as the first argument, one could demand that the agent attempt to clean up after *all* other agents and attempt to prevent *all* safety violations. For more refined behavior, other classes (besides `self` and `everyone`) could be defined.

Our suggestion is problematic for several reasons. (1) Since the agent has no representation of the goals that *other* users are trying to accomplish, it might try to enforce a generalized `restore` constraint with tidying actions that directly conflict with desired goals. In addition, there is the question of when the agent should consider the human "finished" — without an adequate method, the agent could tidy up while the human is still actively working. (2) More generally, the human interface issues are complex — we conjecture that users would find vigilance extremely annoying. (3) Given a complex world where the agent does not have complete information, *any* any attempt to formalize the second half of Asimov's First Law is fraught with difficulties. The agent might reject direct requests to perform useful work in favor of spending *all* of its time sensing to see if some dangerous activity *might* be happening that it *might* be able to prevent.

Conclusion

This paper explores the fundamental question originally posed by Asimov: how do we stop our artifacts from causing us harm in the process of obeying our orders? This question becomes increasingly pressing as we develop more powerful, complex, and autonomous artifacts such as robots and softbots (Etzioni, Lesh, & Segal 1993; Etzioni 1993). Since the positronic brain envisioned by Asimov is not yet within our grasp, we adopt the familiar classical planning framework. To facilitate progress, we focused on two well-defined primitives that capture aspects of the problem: `dont-disturb` and `restore`. We showed that the well-understood, and computational tractable, mechanism of threat detection can be extended to avoid harm.

Other researchers have considered related questions. A precursor of `dont-disturb` is discussed in the work

[9] Formally, serializability (Korf 1987) means that there exists a ordering among the subgoals which allows each to be solved in turn without backtracking over past progress. Trivial serializability means that *every* subgoal ordering allows monotonic progress (Barrett & Weld 1993). While goal ordering is often important among the top level goals, we observe that cleanup goals are usually trivially serializable once the block of top level goals has been solved. For example, the goal of printing a file and the constraint of restoring files to their compressed state are serializable. And the serialization ordering places the printing goal first and the cleanup goal last. As long as the planner considers the goals in this order, it is guaranteed to find the obvious `uncompress-print-compress` plan.

of Wilensky and more extensively by Luria (Luria 1988) under the heading of "goal conflict." Similarly, a precursor of `restore` is mentioned briefly in Hammond *et. al*'s analysis of "stabilization" under the heading of "clean up plans" (Hammond, Converse, & Grass 1992). Our advances include precise and unified semantics for the notions, a mechanism for incorporating `dont-disturb` and `restore` into standard planning algorithms, and an analysis of the computational complexity of enforcing safety and tidiness.

Even so, our work raises more questions than it answers. Are constraints like `dont-disturb` and `restore` the "right" way to represent harm to an agent? How does agent safety relate to the more general software safety (Leveson 1986)? Can we handle tradeoffs short of using expensive decision theoretic techniques? What guarantees can one provide on resource usage? Most importantly, how do we weaken the assumptions of a static world and complete information?

References

Allen, J. 1991. Planning as temporal reasoning. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 3–14.

Asimov, I. 1942. Runaround. *Astounding Science Fiction.*

Barrett, A., and Weld, D. 1993. Characterizing subgoal interactions for planning. In *Proc. 13th Int. Joint Conf. on A.I.*, 1388–1393.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Davis, E. 1990. *Representations of Commonsense Knowledge.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Dean, T., Firby, J., and Miller, D. 1988. Hierarchical planning involving deadlines, travel times, and resources. *Computational Intelligence* 4(4):381–398.

Drummond, M. 1989. Situated control rules. In *Proceedings of the First International Conference on Knowledge Representation and Reasoning.*

Etzioni, O., Lesh, N., and Segal, R. 1993. Building softbots for UNIX (preliminary report). Technical Report 93-09-01, University of Washington. Available via anonymous FTP from `pub/etzioni/softbots/` at `cs.washington.edu`.

Etzioni, O. 1991. Embedding decision-analytic control in a learning architecture. *Artificial Intelligence* 49(1–3):129–160.

Etzioni, O. 1993. Intelligence without robots (a reply to brooks). *AI Magazine* 14(4). Available via anonymous FTP from `pub/etzioni/softbots/` at `cs.washington.edu`.

Fox, M., and Smith, S. 1984. ISIS — a knowldges-based system for factory scheduling. *Expert Systems* 1(1):25–49.

Haddawy, P., and Hanks, S. 1992. Representations for Decision-Theoretic Planning: Utility Functions for Dealine Goals. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning.*

Hammond, K., Converse, T., and Grass, J. 1992. The stabilization of environments. *Artificial Intelligence.* To appear.

Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

Leveson, N. G. 1986. Software safety: Why, what, and how. *ACM Computing Surveys* 18(2):125–163.

Levesque, H., and Brachman, R. 1985. A fundamental tradeoff in knowledge representation. In Brachman, R., and Levesque, H., eds., *Readings in Knowledge Representation.* San Mateo, CA: Morgan Kaufmann. 42–70.

Luria, M. 1988. *Knowledge Intensive Planning.* Ph.D. Dissertation, UC Berkeley. Available as technical report UCB/CSD 88/433.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, 634–639.

Pednault, E. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4(4):356–372.

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, 324–332.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114. Available via FTP from `pub/ai/` at `cs.washington.edu`.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proc. 12th Nat. Conf. on A.I.*

Pollack, M. 1992. The uses of plans. *Artificial Intelligence* 57(1).

Russell, S., and Wefald, E. 1991. *Do the Right Thing.* Cambridge, MA: MIT Press.

Shoham, Y. 1988. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence.* Cambridge, MA: MIT Press.

Tate, A. 1977. Generating project networks. In *Proc. 5th Int. Joint Conf. on A.I.*, 888–893.

Wellman, M., and Doyle, J. 1992. Modular utility representation for decision theoretic planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, 236–242.

Wilkins, D. E. 1988. *Practical Planning.* San Mateo, CA: Morgan Kaufmann.

Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *Proc. 2nd Int. Conf. on A.I. Planning Systems.*