Deductive Planning with Inductive Loops

Martin Magnusson and **Patrick Doherty**
Department of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden
marma@ida.liu.se, patdo@ida.liu.se

Abstract

Agents plan to achieve and maintain goals. Maintenance that requires continuous action excludes the representation of plans as finite sequences of actions. If there is no upper bound on the number of actions, a simple list of actions would be infinitely long. Instead, a compact representation requires some form of looping construct. We look at a specific temporally extended maintenance goal, multiple target video surveillance, and formalize it in Temporal Action Logic. The logic's representation of time as the natural numbers suggests using mathematical induction to deductively plan to satisfy temporally extended goals. Such planning makes use of a sound and useful, but incomplete, induction rule that compactly represents the solution as a recursive fixpoint formula. Two heuristic rules overcome the problem of identifying a sufficiently strong induction hypothesis and enable an automated solution to the surveillance problem that satisfies the goal indefinitely.

Introduction

Research in cognitive robotics has produced autonomous robots such as unmanned aerial vehicles (UAVs) that can carry out complex tasks without human intervention (Doherty 2004; Doherty *et al.* 2004; Doherty &Rudol 2007). But many natural applications are associated with automated planning problems that are beyond the representational capabilities of most automated planners. While one could circumvent this problem by providing ready-made plans for these problems, doing so sacrifices flexibility since such plans will not work in unforeseen circumstances.

Consider e.g. a surveillance task. The goal is to continually get information or video footage of a number of target locations. Such a goal is not achieved, but rather maintained indefinitely or until the surveillance task is aborted. Viewing plans as sequences of actions would result in an infinite number of actions, flying back and forth to keep information on the targets current. Any workable solution must instead view plans as simple programs that make use of some form of loop construct. As a consequence, planning must correspond to a simple form of program synthesis.

Program synthesis is most often considered in a deductive framework as the result of a constructive proof of the program's specification, or the planning goal. Previous research (Magnusson 2007) has investigated deductive planning in Temporal Action Logic (TAL) (Doherty &Kvarnström 2007) where time is explicitly represented by the natural numbers. Achievement goals are then expressed as the existence of a time point at which the goals are satisfied. Maintenance goals, in contrast, must hold at all time points:

$$\forall t\,[\mathrm{Goals}(t)]$$

This formulation strongly suggests the use of mathematical induction over the natural number time line. The base case proof would result in initialization actions that form a set-up phase in the task's solution plan. This would be followed by an induction step proof to produce actions that must be performed iteratively. Finally, applying the principle of mathematical induction would append an infinite chain of copies of the iterated actions to the initialization actions, thereby closing the loop.

However, some difficulties must be overcome before this intuition can be realized. First, mathematical induction usually proceeds in one step increments. One proves that if the proposition holds for x, it will hold for $x+1$. This is inconvenient when multiple actions with non-unit durations are involved in the induction step proof. We solve this problem by using an induction rule parameterized on the induction step size s. The choice of a value for the step size s is delayed until the proof of the induction step is completed.

Second, even if a goal formula is provable using induction, it might not be inductive by itself (Manna &Waldinger 1987). In other words, assuming that the formula holds at x might not provide sufficient grounds for a proof that it holds at $x+s$. Instead, one needs to find a "stronger" induction hypothesis that *is* inductive. The proof of the stronger formula then entails the original formula. Finding an inductive variant of the goal formula is a serious difficulty, and the usual solution is to delegate this task to a human. We add heuristic *goal strengthening* rules that guide the proof process into trying promising strengthened variants of the original formula.

Third, the resulting plan contains a loop and loops can not in general be expressed in first-order logic. We add a greatest fixpoint operator to our logic that admits a compact representation of loops in terms of recursive expressions that can be executed by the agent.

After briefly surveying related work and introducing the above paraphernalia, we present a solution to the surveillance task produced by an automated theorem prover based on natural deduction.

Related Work

While most automated planning systems do not have the capability to generate plans with loops, nor do they have plan representation languages expressive enough to encode solution plans with loops, we discuss some notable exceptions below.

One such exception is Manna and Waldinger's (1987) plan theory. It is based on the Situation Calculus and uses induction over well-founded relations, such as the blocksworld relation *above*(x, y). They apply an automated tableau theorem prover to the task of clearing a block, i.e. ensuring that no block is above the cleared block. A block can be cleared e.g. by clearing the block that is directly above, and then moving that block to the table. In constructing this plan Manna and Waldinger end up with two conditions that are unprovable without a strengthening of their induction hypothesis. However, suggestions for how such strengthening could be automated are not provided.

Cresswell, Smaill, and Richardson (2000) extend previous deductive planning approaches based on intuitionistic linear logic. A recursive data type is introduced along with an induction rule for it. This is linked to the deductive synthesis of recursive plans using concepts from type theory. Automation is provided in Lolli (Hodas &Miller 1994), a logic programming language based on intuitionistic linear logic, where heuristics help at "difficult choice points" such as when choosing what induction rule to use. One step in their proof requires an induction hypothesis strengthening, generalizing their goal of inverting a blocksworld tower, which they state is "amenable to automation". Whether they *do* automate it is not made clear by the paper.

Stephan and Biundo (1993) present a deductive planning framework based on Dynamic Logic. Although they treat "recursively defined notions", again citing the blocksworld *above* relation, to the best of our knowledge they do not generate solution plans with recursion or loops.

In later work (Stephan &Biundo 1996) they introduce an interval-based modal temporal logic called Temporal Planning Logic, which can be used to formulate plans that contain recursive procedures. However, these are "prefabricated general solutions [that] are adapted to special problems". There is no attempt to generate such plans since they claim that this "cannot be carried out in a fully automatic way". If they are taken to mean that there can be no *complete* algorithm for generating recursive procedures it is clear that they are correct since plans with recursion are Turing complete. However this should not stop us from attempting to develop fully automated synthesis for interesting special cases that are useful for large classes of real world problems.

Similarly, Koehler's PHI planner (Koehler 1996) can work with plans that contain complex control structures like conditionals and loops. But loop constructs are reused or modified from existing plans, not generated from scratch.

Bundy et. al. (2006) argue that "induction is needed whenever some form of repetition is required". They present algorithms for constructing new induction rules on the fly instead of relying on a fixed set of predefined rules. Their results are applied to the problem of automatic program synthesis from a specification rather than to the related problem of planning robot actions.

A non-inductive strategy is pursued by Levesque (2005) in his KPLANNER. It explores an interesting middle ground between our direct synthesis of loops and "traditional noniterative planning". The planner works by placing an upper bound on the *planning parameter*, e.g. the time line in our surveillance example. The bounded problem can be solved using classical planning techniques. Next, the planner identifies action sequences that could correspond to unwinded loops and winds them up so that the result is a compact plan with loops. Levesque shows how this method is guaranteed correct for certain classes of planning problems, though it seems to us that an inductive formulation of planning with loops is more straightforward.

More different still are systems based on model checking, such as the Cooperative Intelligent Real-Time Control Architecture (CIRCA) (Musliner, Durfee, &Shin 1993) and the MBP planner (Cimatti *et al.* 2003). While they allow cyclic solutions, they also rely on an explicit state space representation, which we want to avoid. Even if manageable state spaces can be constructed for many benchmark planning problems, we would like to consider planning in a more general setting. An autonomous agent trying to satisfy a goal will have a large body of background knowledge about the world. An explicit representation of the resulting state space would result in a state explosion. One may attempt to circumvent this problem by planning in a limited state representation involving only those fluents that are relevant to the satisfaction of the given goal. But this begs the difficult question of how one would know what fluents are relevant *before* the search for a solution has even begun.

Inductive Proof

Manna and Waldinger (1987) state that "In proving a given theorem by induction, it is often necessary to prove a stronger, more general theorem, so as to have the benefit of a stronger induction hypothesis". Pnueli (2006) provides a simple example:

$$\forall x\, [\exists y\, [1 + 3 + 5 + \cdots + (2x - 1) = y^2]] \tag{1}$$

Proving Formula 1 using induction over the natural numbers is not possible by using the body of the universal quantifier as the induction hypothesis. Instead one must find a stronger *inductive* hypothesis, the proof of which entails the original formula.

Now, consider the surveillance task mentioned above. Suppose that we need to have up-to-date image data, at most 15 minutes old, from three locations. Our UAV continually records video so visiting a location suffices to obtain image data from it. We state that the location feature loc for a UAV uav_1 assumes, e.g., the value $\mathrm{pos}(160, 500)$ at some time point y by *value*$(y, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 500)$. If time points represent seconds then we can express the fact that the visit takes place no more than 15 minutes from the current time point x by $0 \leq y - x < 900$. In the surveillance task the goal is to *always* have recent information, i.e. the above condition is satisfied at all time points:

$$\forall x\,[\exists y\,[\textit{value}(y, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 500) \wedge 0 \leq y - x < 900]] \quad (2)$$

Formula 2, unfortunately, is too weak as an induction hypothesis for the same reason as Formula 1. Assuming only the *existence* of a visiting time y within 900 seconds from x does not leave any time for flying around while still making sure the condition holds at $x + s$. In the worst case, 899 seconds have already passed since y and the UAV would need to visit $\mathrm{pos}(160, 500)$ again the very next second.

The task of finding a strengthening of the goal formula is often left to a human. E.g., Pnueli's Formula 1 is easily proved if the existential quantifier is removed and the variable y replaced by the variable x. This stronger formula entails the original by existential generalization. But an autonomous UAV cannot depend on human ingenuity to help solve its planning problems. Moreover, as previously mentioned, there can be no complete proof method for induction in general. Our solution is instead the addition of goal strengthening rules. These are proof rules that suggest stronger versions of goals of certain commonly occurring forms. Their theoretical redundancy but practical significance, in guiding the proof search, make these rules heuristic in nature.

Regularity Heuristic

If the robot knew the time point at which a location was last visited it would have more freedom in planning its next visit. E.g., if the UAV decides on a regular flight schedule it would, at any given time point, know how much time had passed since its last visit to the target location. We can capture this intuition as a heuristic goal strengthening rule. The following rule guides a backward chaining proof search, and should therefore be read backwards from the conclusion to the premise. If we are trying to prove that there always exists a time point y within n time points where P holds, then we might instead try to prove that there is some regular schedule of length s no larger than n such that P always holds for each repetition i of the schedule, with some offset m less than n:

(*Regularity Heuristic*)

$$\frac{\exists s\,[s \leq n \wedge \exists m \forall i\,[P(is + m) \wedge 0 \leq m < n]]}{\forall x \exists y\,[P(y) \wedge 0 \leq y - x < n]}$$

Theorem 1. *The regularity heuristic is sound.*

Proof. Assume the premise and that x is an arbitrary number. Choose the smallest i such that $is + m \geq x$. By assumption $P(is + m)$ holds, and since $s \leq n$ and $m < n$ the distance from x to $is + m$ must be less than n. Thus for any x there exists a y within n time points (namely $is + m$) such that $P(y)$ holds. □

Applying this rule to Formula 2 produces a new goal:

$$\exists s\,[s \leq 900 \wedge \exists m\,[\forall i\,[\textit{value}(is + m, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 500) \wedge 0 \leq m < 900]]]$$

Synchronization Heuristic

Of course, as long as there is only one location to monitor, the UAV could get by with the trivial solution of just hovering over it. More interesting behaviour is necessary if up-to-date information from several different locations is required:

$$\begin{aligned}
&\exists s\,[s \leq 900 \wedge \\
&\quad \exists m\,[\forall i\,[\textit{value}(is + m, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 500) \wedge \\
&\qquad 0 \leq m < 900]]] \wedge \\
&\exists s\,[s \leq 900 \wedge \\
&\quad \exists m\,[\forall i\,[\textit{value}(is + m, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 680) \wedge \\
&\qquad 0 \leq m < 900]]] \wedge \\
&\exists s\,[s \leq 900 \wedge \\
&\quad \exists m\,[\forall i\,[\textit{value}(is + m, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(400, 680) \wedge \\
&\qquad 0 \leq m < 900]]]
\end{aligned} \quad (3)$$

While each sub-goal could be planned for separately, this would most likely result in a very inefficient proof search. The UAV would first construct plans that work for each location in isolation and backtrack if the combined plan requires it to visit two locations simultaneously, again reconsidering each plan in isolation. A better strategy would be to assume that all of the visits are part of the *same* schedule and consider them simultaneously. This strategy corresponds to another goal strengthening heuristic. If trying to prove the existence of several different schedules with independent iteration variables we may instead synchronize them into one schedule and iteration:

(*Synchronization Heuristic*)

$$\frac{\exists s \forall i\,[P_1(s, i) \wedge \cdots \wedge P_n(s, i)]}{\exists s_1 \forall i_1\,[P_1(s_1, i_1)] \wedge \cdots \wedge \exists s_n \forall i_n\,[P_n(s_n, i_n)]}$$

Theorem 2. *The synchronization heuristic is sound.*

Proof. Assume the premise. Distribute the universal quantifier over the conjunction. Since the entire conjunction holds for some s, each conjunct must hold for some s, as in the conclusion. □

By moving the m variables of Formula 3 into the prefix, renaming them apart, and applying the above heuristic, we arrive at a new proof goal:

$$\begin{aligned}
&\exists m_1, m_2, m_3, s\,[\forall i\,[\\
&\quad s \leq 900 \wedge \\
&\quad \textit{value}(is + m_1, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 500) \wedge \\
&\quad 0 \leq m_1 < 900 \wedge \\
&\quad \textit{value}(is + m_2, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(160, 680) \wedge \\
&\quad 0 \leq m_2 < 900 \wedge \\
&\quad \textit{value}(is + m_3, \mathrm{loc}(\mathrm{uav}_1)) = \mathrm{pos}(400, 680) \wedge \\
&\quad 0 \leq m_3 < 900]]
\end{aligned} \quad (4)$$

Induction Rule

The goal strengthening heuristics enable the formulation of an induction rule for planning with loops by introducing initialization actions I in the base case and loop step actions A in the induction step, as further explained below:

(*Induction Rule*)

$$\frac{\begin{array}{l} I \rightarrow P(0) \\ \forall x\,[P(x) \wedge A(x) \rightarrow P(x+s)] \end{array}}{I \wedge \nu X(x)[A(x) \wedge X(x+s)](0) \rightarrow \forall i\,[P(is)]}$$

The initialization actions I make sure that the induction base case holds. The actions A inside the loop depend on the induction variable x and make sure the goal P holds for the next loop iteration at $x + s$.

When the induction principle is applied the loop step actions need to be repeated indefinitely. This could be expressed in a declarative manner using a universal quantifier. But since the purpose of the formula is to function as an executable plan for the UAV, we express it in a constructive manner by introducing a greatest fixpoint operator ν. The expression $\nu X(x)[A(x) \wedge X(x+s)]$ defines a new relation X with a time point argument x at which the loop step actions A hold. Moreover, X itself holds at $x + s$, thereby setting up a recursive iteration. Finally, the loop expression is initialized by providing the base case time point 0 as its argument. Note that we assume $s > 0$ and $i \geq 0$ since s corresponds to the loop duration and i to loop repetitions, none of which make sense for negative values.

The fixpoint loop together with the initialization actions ensure only that P holds at any multiple i of s. However, the goal strengthening heuristics have already put the surveillance goal into this form in Formula 4. The original goal, covering all time points as in Formula 2, is a guaranteed consequence due to the heuristics' soundness. Soundness of the induction rule will be proved using the following lemma.

Lemma 3. *The induction rule fixpoint formula corresponds to occurrences of actions A at regular intervals starting at 0 and spaced by s, i.e.:*

$$\begin{array}{l} \nu X(x)[A(x) \wedge X(x+s)](0) \Leftrightarrow \\ A(0) \wedge A(s) \wedge A(2s) \wedge \cdots \end{array}$$

Proof. The Knaster-Tarski fixpoint theorem states the existence of a greatest fixpoint of a formula F, defined by Kleene's iterative construction as the limit of the generalized conjunction bounded by ω:

$$\nu X(x)[F(X(x))] \Leftrightarrow \bigwedge_{i<\omega} F^i(\text{true})$$

where F^0 is the identity function and $F^{(i+1)}$ is F with occurrences of $X(x)$ replaced by F^i instantiated with x. In our case we have $F = A(x) \wedge X(x+s)$ and:

$$\begin{array}{l} F^0 = \text{true} \\ F^1 = F(F^0) = A(x) \wedge \text{true} \\ F^2 = F(F^1) = A(x) \wedge A(x+s) \\ F^3 = F(F^2) = A(x) \wedge A(x+s) \wedge A(x+s+s) \\ \vdots \end{array}$$

We see that the form of F^i is:

$$F^i = A(x+0s) \wedge A(x+1s) \wedge \cdots \wedge A(x+is)$$

Clearly F^i subsumes $F^{(i-1)}$ for every i in the generalized conjunction above. What remains is then F^ω, which we write in the form of an infinite conjunction:

$$F^\omega = A(x) \wedge A(x+s) \wedge A(x+2s) \wedge \cdots$$

By instantiating the free variable x with 0 we arrive at the right hand side of Lemma 3:

$$A(0) \wedge A(s) \wedge A(2s) \wedge \cdots$$

□

Theorem 4. *The induction rule is sound.*

Proof. The proof that the induction rule conclusion follows from the premises proceeds by mathematical induction over the size k of the domain $\{0, \ldots, k\}$ of the natural number i.

In the base case $i = \{0\}$. Given that the premises hold, assume the antecedent of the conclusion implication, including I. This, together with the first premise, immediately produces $P(0)$ which can be rewritten $P(0s)$. Since 0 is the only value i may assume we have $P(is)$ for all i.

As induction hypothesis we assume that the induction rule is sound for $i = \{0, \ldots, k\}$ and show that it holds for $i = \{0, \ldots, k, k+1\}$. Thus the conclusion follows from the premises for all i up to and including k and what remains is to show that it follows for $i = k + 1$, i.e. to show that $I \wedge \nu X(x)[A(x) \wedge X(x+s)](0) \rightarrow P((k+1)s)$. Assume the antecedent. The induction hypothesis holds for $i = k$ and gives us $P(ks)$. According to Lemma 3 we have $A(0) \wedge A(s) \wedge A(2s) \wedge \cdots \wedge A(ks) \wedge \cdots$. Taken together we have $P(ks) \wedge A(ks)$ which, from the second induction rule premise, produces $P(ks + s)$, i.e. $P((k+1)s)$. We have now proved the consequent of the conclusion implication for all possible values of i and the universally quantified conclusion follows.

By the principle of mathematical induction the rule holds for all possible sizes of the domain of i. □

Bundy's Criteria for Heuristic Rules

While the heuristics and induction rule can not be complete, one would wish for them to cover some useful and commonly occurring cases. At the same time, one wants to avoid creating a set of specialized ad hoc rules that solve only a particular problem. Bundy (1988) considers desirable properties of *proof plans,* which are used to guide the search for a proof. Our heuristic rules are clearly related to the concept of proof plans, and evaluating them with regard to Bundy's properties can help explain their usefulness. Specifically, Bundy's generality and expectancy (why one expects the rule to work) properties are relevant and we examine each of the rules with respect to these properties below.

Regularity Heuristic The regularity heuristic is applicable to any condition P for which only a limited amount of time n should pass before it is satisfied again. The heuristic suggests that these can be accomplished by deciding on a fixed period s and offset m at which to satisfy them. Real life examples include e.g. a bus servicing a stop every 20 minutes, or backing up data each week on Friday.

Synchronization Heuristic The synchronization heuristic is applicable when several iteration periods s_i are involved and suggests synchronizing them with each other by using a common period s. This is especially useful when the same resource is involved in the different tasks. E.g., if the UAV's surveillance of each location is planned in isolation there is great risk that the different schedules will conflict when merged. By synchronizing the iteration periods and organizing all the tasks in a single schedule one readily detects and resolves attempts of conflicting resource use.

Induction Rule The heuristics taken together with the induction rule provide the high-level structure for solving a class of commonly occurring temporally extended planning goals by planning actions in an iterative loop. Specifically, any tasks where some general constraint is satisfied by periodic actions are candidates. Such tasks are plentiful, with examples from different domains such as autonomous surveillance or a housekeeper robot watering the plants every day to keep them alive. In fact, much of your day probably consists of loops that are designed to keep some parameter within an acceptable range through regular action, e.g. mental alertness through regular sleep and brain caffeination.

Automating Planning with Loops

The question of automation still remains. Previous work on deductive planning in Temporal Action Logic used a compilation of TAL formulas into Prolog programs (Magnusson 2007). But Prolog's limited expressivity makes it inadequate for our present purposes. Instead, our current work utilizes Pollock's natural deduction with *quantifier-free form* (Pollock 1999) that replaces the somewhat cumbersome quantifier introduction and elimination rules found in most natural deduction systems with unification.

Our theorem prover implementation is named ANDI, for augmented natural deduction intelligence. Its rules are divided into *forward* and *backward* rules. Forward rules are triggered whenever possible and are designed to converge on a stable set of conclusions to avoid generating new inferences forever. Backward rules, in contrast, are used to search backwards from the current proof goal and thus exhibit goal direction. Combined, the result is a bi-directional search for proofs.

We make non-monotonic reasoning and planning possible through a "natural abduction" proof rule. Relations from a set of *abducibles* are allowed to be assumed rather than proven, as long as doing so does not lead to inconsistency. It is well known that the problem of determining consistency of a first-order theory is not even semi-decidable. Our theorem prover relies on its forward rules to implement an *incomplete* consistency check. If an abductive assumption and the triggering forward rules produce a contradiction, then the prover needs to backtrack and cancel some assumption that participates in the proof of the contradiction. If an inconsistent assumption remains undetected the resulting implication for a goal formula P would have the form $\perp \rightarrow P$ and thus, while consistent, be tautological and void.

The Surveillance Problem

Given the heuristics and the induction rule, the only thing that ANDI is missing to solve the surveillance problem is a formalization of an action for flying between locations. A version that takes the distance between the locations into account is presented below. Note that the universal quantifiers for agents, time, and locations are dropped in quantifier-free form, and the corresponding variables a, t, l_1, and l_2 are prefixed by question marks:

```
(-> (and (= (value ?t (loc ?a)) ?l1)
         (occurs ?a ?t (fly ?l1 ?l2)))
    (= (value (+ ?t (dist ?l1 ?l2))
              (loc ?a))
       ?l2))
```

The surveillance goals of the form (2) are put into quantifier-free form through skolemization. Existentially quantified variables are replaced by skolem constants and functions, prefixed by a \$. E.g., since the existentially quantified y variables depend on the universally quantified x variables they result in skolem functions of the form `($y ?x)`. The goal is then:

```
(and (= (value ($y1 ?x1) (loc uav1))
        (pos 160 500))
     (<= 0 (- ($y1 ?x1) ?x1))
     (< (- ($y1 ?x1) ?x1) 900)
     (= (value ($y2 ?x2) (loc uav1))
        (pos 160 680))
     (<= 0 (- ($y2 ?x2) ?x2))
     (< (- ($y2 ?x2) ?x2) 900)
     (= (value ($y3 ?x3) (loc uav1))
        (pos 400 680))
     (<= 0 (- ($y3 ?x3) ?x3))
     (< (- ($y3 ?x3) ?x3) 900))
```

Finally, we need to know the initial location of the UAV:

```
(= (value 0 (loc uav1)) (pos 160 500))
```

Provided this input, the ANDI theorem prover automatically generates the following plan:

```
(and (occurs uav1 0 (fly (pos 160 500)
                         (pos 160 680)))
     (occurs uav1 180 (fly (pos 160 680)
                           (pos 400 680)))
     ((gfp (rel ?x)
       (and (occurs uav1 (+ ?x 420)
                     (fly (pos 400 680)
                          (pos 160 500)))
            (occurs uav1 (+ ?x 720)
                     (fly (pos 160 500)
                          (pos 160 680)))
            (occurs uav1 (+ ?x 900)
                     (fly (pos 160 680)
                          (pos 400 680)))
            (rel (+ ?x 720)))) 0))
```

Number of Targets	CPU Time in Seconds
10	1
20	5
30	17

Table 1: Time spent by ANDI (on average over three runs) solving different size surveillance problems on a Pentium M 1.8 GHz laptop with 512 MB of RAM.

ANDI integrates constraint solvers and rewrite rules into the proof process. In this case an inequality constraint solver determined the value 720 for s and rewrite rules were used e.g. to simplify instantiated arithmetic expressions, most importantly the distance function between location coordinates. Very large (or infinite) domains such as numeric time and coordinate spaces are no problem for the backward chaining search, which picks relevant values from the huge set of possibilities by unification with the goal or a sub-goal derived from the goal. Moreover, the most general unifier does not necessarily instantiate variables and can thereby often avoid early commitment to choices that might lead to unnecessary backtracking. All of these factors contribute to the theorem prover's practical usefulness in solving real world size planning problems, as exemplified by Table 1.

Conclusion

Many interesting problems require planning with loops, for which there can be no complete algorithm. We considered a useful subset of temporally extended goals that can be solved by regular repeated action. These problems can be solved using a proof rule for mathematical induction together with a couple of heuristic rules that put the goal into a form where the induction rule is applicable. Natural deduction conveniently captures this relatively complex proof strategy through its extensible set of proof rules since, when a planning goal matches a rule's conclusion, the rule's premises functions as a proof agenda to follow. For the induction rule, the result is a fixpoint loop solution that compactly represents an infinite plan.

We applied the methodology to a UAV surveillance problem that was quickly solved by an automated natural deduction theorem prover. By concentrating on surveillance we could ignore, for now, the frame problem since no reasoning about persistence was involved. Furthermore, the temporally extended goal was solved by a simple application of mathematical induction over the TAL time line since it consists of the natural numbers. However, there are clearly more complex classes of planning problems, requiring solutions of an iterative nature, where the same methodology could be attempted. Consider e.g. a logistics mission where the goal is to distribute supply crates. The crates' current locations and target destinations could be represented as well-founded sets, thereby making induction applicable. A plan that loops over crates would represent a constant size solution, regardless of the number of crates that need to be distributed. We believe that this ability to utilize complex proof rules, such as induction, to find compact plans in highly expressive formalisms, such as loops in fixpoint logic, provides logical planning with a potential to scale up to realistically sized problems.

Acknowledgements

We thank Andrzej Szalas for fruitful discussions about fixpoints.

This work is supported in part by the Swedish Research Council VR grant 2005-3642, the National Aeronautics Research Program NFFP04 S4203, CENIIT, and the Strategic Research Center MOVIII, funded by the Swedish Foundation for Strategic Research, SSF.

References

Bundy, A.; Dixon, L.; Gow, J.; and Fleuriot, J. D. 2006. Constructing induction rules for deductive synthesis proofs. *Electronic Notes in Theoretical Computer Science* 153(1):3–21.

Bundy, A. 1988. The use of explicit plans to guide inductive proofs. In *Conference on Automated Deduction*, 111–120.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Cresswell, S.; Smaill, A.; and Richardson, J. 2000. Deductive synthesis of recursive plans in linear logic. In *Proceedings of the 5th European Conference on Planning (ECP '99)*, 252–264. London, UK: Springer-Verlag.

Doherty, P., and Kvarnström, J. 2007. Temporal action logics. In Lifschitz, V.; van Harmelen, F.; and Porter, B., eds., *Handbook of Knowledge Representation*. Elsevier.

Doherty, P., and Rudol, P. 2007. A UAV search and rescue scenario with human body detection and geolocalization. In *20th Australian Joint Conference on Artificial Intelligence (AI07)*.

Doherty, P.; Haslum, P.; Heintz, F.; Merz, T.; Persson, T.; and Wingman, B. 2004. A distributed architecture for intelligent unmanned aerial vehicle experimentation. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 221–230.

Doherty, P. 2004. Advanced research with autonomous unmanned aerial vehicles. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*, 731–732.

Hodas, J. S., and Miller, D. 1994. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation* 110(2):327–365.

Koehler, J. 1996. Planning from second principles. *Artificial Intelligence* 87(1-2):145–186.

Levesque, H. J. 2005. Planning with loops. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, 509–515.

Magnusson, M. 2007. *Deductive Planning and Composite Actions in Temporal Action Logic*. Licentiate thesis, Linköping University. http:

//www.martinmagnusson.com/publications/magnusson-2007-lic.pdf.

Manna, Z., and Waldinger, R. 1987. How to clear a block: A theory of plans. *Journal of Automated Reasoning* 3(4):343–377.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: A cooperative intelligent real time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6):1561–1574.

Pnueli, A. 2006. Analysis of reactive systems: Lecture 7. http://cs.nyu.edu/courses/spring06/G22.3033-005/lecture7.pdf. Visited Feb. 2008.

Pollock, J. 1999. Natural deduction. Technical report, Department of Philosophy, University of Arizona. http://www.sambabike.org/ftp/OSCAR-web-page/PAPERS/Natural-Deduction.pdf.

Stephan, W., and Biundo, S. 1993. A new logical framework for deductive planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 32–38.

Stephan, W., and Biundo, S. 1996. Deduction-based refinement planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 213–220.