Some Complexity and Correctness Results for Proof Planning

Toby Walsh[1]

University of York, York, England. tw@cs.york.ac.uk

Abstract. We prove that determining if a valid proof plan exists is undecidable in general, even if the meta-theory in which pre- and post-conditions are checked is itself decidable. On a more positive note, we prove that an iterative deepening proof planner is sound (only finds valid proof plans) and complete (if it terminates signaling failure or fails to terminate then no valid proof plan exists). We also show that a depth-first proof planner is sound but not complete as there exist valid proof plans that it will not find.

1 Introduction

In recent years, researchers have started to study planning from a more theoretical perspective [Byl94,BN95]. The outcomes of this research programme include complexity results about the difficulty of planning under various assumptions. Whilst planning with operators using a first-order language is undecidable in general, more tractable cases have been identified. For example, planning with STRIPS operators restricted to a propositional language is PSPACE-complete. It is therefore timely to consider how such results might be mapped onto proof planning.

2 Formal background

Proof planning was introduced by Bundy in [Bun88] and implemented within the CLAM system [BvHHS90]. A **proof plan** is a tree of methods. A proof plan is **valid** iff each method within it is applicable and the leaf nodes are terminating methods. We say that it is **invalid** otherwise. A method is **applicable** iff it matches the current (sub)goal, and both its pre- and post-conditions hold. The children of each applicable method inherit the output goals of their parent method as input goals. A method is **terminating** iff it has no output goals. Terminating methods can only occur at the leaf nodes. Associated with each method is a tactic of the same name. The tactic attempts to build the object level proof associated with the method. Tactics are not, however, guaranteed to succeed.

We define Proof Plan Existence as the problem of determining if a valid proof plan exists. We define Proof Plan Validity as the problem of determining if a given proof plan is valid. Note that Proof Plan Validity does

not necessarily imply that the goal is an object level theorem. The tactics associated with each method are not guaranteed to succeed. The complexity of proof planning can therefore differ from the complexity of theorem proving in the underlying object level logic.

Recently we have built a simple proof planner, called CLAM-Lite. Our aim is to add proof planning capabilities to the Maple computer algebra system. However, one positive side-effect is that we now have a simple enough proof planner that we can start to reason about it formally. The depth-first proof planner is given in Table 1 (other search strategies like iterative-deepening can be defined in a similar fashion), and an example method in Table 2.

```
proof_plan(Goal,[Method|Plan]):-
   apply(Method,Goal,Outputs),
   map_proof_plan(Outputs,Plan).

apply(Method,Goal,Outputs):-
   method(Method,Goal,Pre,Post,Outputs),
   call_goals(Pre),
   call_goals(Post).

map_proof_plan([],[]).
map_proof_plan([Output|Outputs],[Plan|Plans]):-
   proof_plan(Output,Plan),
   map_proof_plan(Outputs,Plans).
```

Table 1. The simple depth-first proof planner used within CLAM-Lite. Other search strategies (e.g. breadth-first, and iterative deepening) are defined in a similar fashion.

```
method(equality(poly_expand,A1,B1),
   H==>A=B,
   [polynomial(A),
    polynomial(B)]
   [normal_form(A,A1),
    normal_form(B,B1)]
   [H==>A1=B1]).
```

Table 2. An example method used within CLAM-Lite. A method has five slots: the method (and tactic) name, the input goal, the pre-conditions, the post-conditions, and the list of output goals. The method above proves equality of two polynomial terms by expanding them out into a normal form. The input goal is a sequent with the set of hypotheses on the left and the conclusion on the right.

3 Complexity results

We first prove that finding proof plans is, in general, undecidable.

Theorem 1. PROOF PLAN EXISTENCE *is undecidable.*

Proof. We construct a set of methods which simulate a Turing machine. We then show that a proof plan is valid iff the Turing machine halts in the accept state. Determining if a valid proof plan exists thus reduces to answering the halting problem and is undecidable. Our initial goal is a triple containing the tape (which can be represented by a list), the position of the head (which can be represented by an integer), and the state of the Turing machine (which can be represented by some finite alphabet). For each transition and state of the Turing machine, we define a method. For example, consider a Turing machine which in state s_0 reads 0 from the tape l_0, replaces 0 by 1, moves the head to the right, and switches to state s_1. Corresponding to this transition, we define a method whose input is the triple $\langle l_0, n, s_0 \rangle$. This method has a single precondition that the n-th element of l is 0. The method's post-conditions construct a new list l_1 from l_0 by replacing the n-th element with 1. The output triple is then $\langle l_1, n+1, s_1 \rangle$. All terminating methods correspond to halt instructions which leave the Turing machine in the accept state. Such methods return the empty output goal. A valid proof plan is therefore a sequence of methods which end in a terminating method iff the Turing machine halts in the accept state.

This is bad news but is perhaps to be expected as planning is undecidable in general. However, the situation is worse than this theorem suggests as even determining if a *given* proof plan is valid is undecidable. The problem is that the meta-theory in which we check the pre- and post-conditions can itself be undecidable.

Theorem 2. PROOF PLAN VALIDITY *is undecidable.*

Proof. We construct a single (terminating) method which is applicable iff a Turing machine halts in the accept state. The method takes as input the description of a Turing machine and a tape. In our meta-theory, we define the semantics for the execution of a Turing machine. The pre-condition of the method then simply checks to see if the Turing machine halts in the accept state when given the particular input tape.

These results are perhaps not too surprising since proof planning can be applied to undecidable meta-theories. Can we obtain more useful results for some special cases? For example, what if we restrict the meta-theory to, say, decidable languages? Unsurprisingly, checking whether a given proof plan is valid is now decidable.

Theorem 3. PROOF PLAN VALIDITY *in a decidable meta-theory is itself decidable.*

Proof. Determining if a method is applicable is decidable since testing if the goal matches against the input slot is decidable, and since testing if the pre- and post-conditions hold is itself decidable. A simple algorithm to recurse over the structure of the proof plan, determining if each method is applicable therefore decides validity.

Unfortunately, deciding if a valid proof plan exists remains undecidable even if the meta-theory in which we are proof planning is itself decidable.

Theorem 4. PROOF PLAN EXISTENCE *can be undecidable even if the associated meta-theory is decidable.*

Proof. The meta-theory used by the set of methods constructed in the proof of Theorem 1 is decidable.

What then of a more tractable case like a propositional meta-theory? As in planning, the following results also apply to first-order theories with finite quantification since all quantified terms can be expanded out to give a purely propositional (and finite) language. Under such restrictions, proof planning has more tractable complexity results.

Theorem 5. PROOF PLAN VALIDITY *in a propositional meta-theory is coNP-complete.*

Proof. We show that determining if a proof plan is invalid, PROOF PLAN INVALIDITY is NP-complete. Hence PROOF PLAN VALIDITY is coNP-complete. Consider a proof plan that is invalid. Our polynomial witness is either a leaf node that is not terminating or a method which is not applicable. A method is not applicable iff its input does not match the current goal or its pre-conditions and post-conditions do not hold[1]. The pre-conditions and post-conditions do not hold if one of the pre- or post-conditions is invalid. That is, if their negation is satisfiable. Hence, our polynomial witness is simply a truth assignment that makes the negation of this pre- or post-condition satisfiable. PROOF PLAN INVALIDITY is therefore in NP. To show it is NP-complete, we map SATISFIABILITY, the satisfiability of a propositional formula φ into PROOF PLAN INVALIDITY as follows. We construct a single terminating method which takes any input and has a single pre-condition $\neg\varphi$. If a proof plan containing this single method is invalid then the pre-condition $\neg\varphi$ is not valid. That is, φ must be satisfiable. Hence we have reduced SAT, which is NP-complete to PROOF PLAN INVALIDITY with a propositional meta-theory.

Finding a valid proof plan is likely to remain intractable in the worst case even when the meta-theory is propositional. Unfortunately, valid proof plans can be exponentially long compared to the size of the input goal and the method base.

[1] Post-conditions are always supposed to succeed but we do not need to insist upon this for this proof.

Theorem 6. *There exist problems in which the shortest valid proof plan is exponential in length despite the meta-theory in which we proof plan being propositional.*

Proof. In fact, we prove that there exist a polynomial number of methods and a polynomial sized input goal for which the shortest valid proof plan is exponential in length. These methods have empty pre- and post-conditions (all computation is done by matching on the inputs and outputs). Alternatively we can place propositional tautologies in the pre-condition slots of these methods. Consider a tower of Hanoi problem with n disks and 3 pegs. Let L_n be the shortest plan to move the n disks from one peg to another. Then $L_1 = 1$ and $L_n = 2 * L_{n-1} + 1$. This has solution $L_n = 2^n - 1$. We now encode this problem into proof planning. The input goal is `table([disk1,disk2,...,diskn],[],[])` representing the fact that all n disks are on the first peg. Their are $O(n^2)$ methods to move disks between pegs. For example, to move `disk1` from the 1st to the 2nd peg, there is a single method:

```
method(move(peg1,peg2),
   H==>table([disk1|Peg1],Peg2,Peg3),
   [],[],
   [H==>table(Peg1,[disk1|Peg2],Peg3)]).
```

As `disk1` is the smallest disk, it can be moved without constraint. For `disk2` we have methods for each of the possible disks it can be placed upon. For example, there is a method to move `disk2` from the 3rd peg onto `disk5` on the 2nd peg:

```
method(move(peg3,peg2),
   H==>table(Peg1,[disk5|Peg2],[disk2|Peg3]),
   [],[],
   [H==>table(Peg1,[disk2,disk5|Peg2],Peg3)]).
```

There are 6 possible directions we can move disks, n possible disks we can pick up and at most $n-1$ acceptable disks to place them upon. Hence there are $O(n^2)$ methods in the method base.

4 Correctness results

We now turn to reasoning about proof planning as implemented within a system like CLAM-Lite. Unfortunately the depth-first planner given in Table 1 is not complete. Depth-first search can be trapped into an infinite descending chain and so may fail to find a valid proof plan. We therefore also consider iterative deepening and breadth-first proof planners as neither are trapped by infinite descending chains. We first prove that there are three possible outcomes of applying the depth-first, iterative deepening or breadth-first proof planners: success and a proof plan is found; failure and the search space is exhausted before a proof plan is returned; or non-termination.

Theorem 7. *The depth-first, iterative deepening and breadth-first proof planners can stop with success, stop with failure, or fail to terminate.*

Proof. Consider a single terminating method with no pre- or post-conditions, and a goal that matches the input formula. All the proof planners immediately apply this method and stop with success.

Consider a goal which does not match any of the input formulae of any of the methods in the method base. All the proof planners immediately fail.

Consider two methods, both with no preconditions one of which transforms the input goal A to B and the other which transforms the input goal B to A. All the proof planners loop on the goal A, trying to build a plan which alternates application of the two methods.

We next prove that the iterative deepening and breadth-first proof planners are correct and complete. That is, if they return a proof plan then it is valid (correctness) and if they terminate signaling failure or fail to halt then no valid proof plan exists (completeness).

Theorem 8. *If the iterative deepening or breadth-first proof planners terminate returning a solution then the solution is a valid proof plan. If they terminate signaling failure or fail to halt then no valid proof plan exists.*

Proof. Correctness is, as is often the case, easier than completeness. The proof of correctness uses induction on the depth of the proof plan. In the base case, the single method applied is terminating. In the step case, we apply a (non-terminating) method and appeal to the induction hypothesis for each of the subgoals.

To prove completeness, suppose a valid proof plan exists. Then it has some maximum depth k. We claim that the kth iteration of the iterative deepening proof planner will find this proof plan. A similar argument can be given for the breadth-first proof planner. Hence if the iterative deepening or breadth-first proof planners fail to find a proof plan then no valid proof plan exists. Consider the first method applied at the root of the proof plan. Now the kth iteration of the iterative deepening proof planner examines all possible methods to apply to the root in turn so will eventually select this method. As the proof plan is valid, its pre- and post-conditions hold. Hence the method is successfully applied and the appropriate subgoals constructed. A similar argument holds for subsequent methods in the proof plan. Hence, the kth iteration of the iterative deepening proof planner finds this proof plan.

We end by proving that the depth-first proof planner is sound but not complete. A valid proof plan may exist but a depth-first proof planner can fail to find it.

Theorem 9. *If the depth-first proof planner terminates returning a solution then the solution is a valid proof plan. If it terminates signaling failure then no valid proof plan exists. However, if it fails to halt then a valid proof plan may or may not exist.*

Proof. The proof is similar to that for the iterative deepening and breadth-first proof planners. The major difference is the proof of the last of the three results. Consider three methods, all without pre- and post-conditions, the first of which transforms the input goal A to B, the second which transforms the input goal B to A, and the third which is terminating and applies to the input goal A. Suppose the methods are listed in the method base in this order. Then the depth-first proof planner will loop given the goal A, trying to build a plan which alternates application of the first two methods. However, a valid proof plan exists which simply applies the third method.

5 Related work

Chapman proved that planning is undecidable [Cha87]. Bylander subsequently studied classes of planning problems which are more tractable [Byl94]. For example, he proved that propositional STRIPS planning is PSPACE-complete. Interestingly, despite the fact that plans in propositional STRIPS planning can be exponentially long, planning is still in PSPACE. As the size of states is bounded, we can guess an intermediate state between our initial and goal states, and find plans from the initial state to this intermediate state, and from the intermediate state to the goal state. Recursively applying this procedure, we can find a plan using space that is just the logarithm of the plan length. Even more severe restrictions are required to guarantee that planning is polynomial or even NP-complete. For example, if the postconditions are only positive then propositional STRIPS planning is monotonic and NP-complete.

6 Conclusions

We have proved that determining if a valid proof plan exists is undecidable in general, even if the meta-theory in which pre- and post-conditions are checked is itself decidable. Even with a propositional meta-theory, valid proof plans can be exponentially long. On a more positive note, we have proved that an iterative deepening proof planner is sound (only finds valid proof plans) and complete (if it terminates signaling failure or fails to terminate then no valid proof plan exists). We have also shown that the depth-first proof planner is sound but not complete. A valid proof plan exists that the depth-first proof planner will fail to find. This proof plan will be found by the iterative deepening or the breadth-first proof planners.

Acknowledgments

The author is supported by an EPSRC advanced research fellowship. He is a member of both the DReaM research group in Edinburgh and the APES cross-university research group. He wishes to thank Axel Schairer for discussion about these complexity results.

References

[BN95] C. Backström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[Bun88] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In R. Luck and R. Overbeek, editors, *CADE9*. Springer-Verlag, 1988. Longer version available as DAI Research Paper No. 349, Dept. of Artificial Intelligence, Edinburgh.

[BvHHS90] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.

[Byl94] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.