Planning and Proof Planning

Erica Melis[1] and **Alan Bundy**[2]

Abstract. The paper adresses proof planning as a specific AI planning. It describes some peculiarities of proof planning and discusses some possible cross-fertilization of planning and proof planning.

1 Introduction

Planning is an established area of Artificial Intelligence (AI) whereas proof planning introduced by Bundy in [2] still lives in its childhood. This means that the development of proof planning needs maturing impulses and the natural questions arise `What can proof planning learn from its Big Brother planning?' and `What are the specific characteristics of the proof planning domain that determine the answer?'.
In turn for planning, the analysis of approaches points to a need of mature techniques for practical planning. Drummond [8], e.g., analyzed approaches with the conclusion that the success of Nonlin, SIPE, and O-Plan in practical planning can be attributed to hierarchical action expansion, the explicit representation of a plan's causal structure, and a very simple form of propositional resource allocation rather than to "precondition achievement" which is the predominant formulation of planning in the AI community. Therefore the techniques of proof planning that succeeded so far might be of interest for other application areas and problem classes.
To provide a feeling, under which conditions approaches or techniques from proof planning can be adopted for planning in realistic environments, we discuss some important features of the proof planning domain. In order to contribute to a cross-fertilization of AI planning and proof planning, we briefly describe lessons that can be learned from planning or proof planning.. This paper extends a description given in [3].

2 Proof Planning

While humans can cope with long and complex proofs and have strategies to avoid less promising proof paths, automated theorem proving suffers from exhaustive search in super-exponential search spaces. Some empirical sources [19, 9] provide evidence that mathematicians use specific methods (e.g. diagonalization), intelligently guide the search for proofs, and plan a proof during the proof discovery process. E.g., the German mathematician Faltings, who proved Mordell's Conjecture, described in [9] that "*We know from experience that certain inferences are usually successful under certain prerequisites. So first we ponder about a reasonable way to proceed to prove the theorem. In other words, we roughly plan: If we get a certain result the next result will follow and then the next etc. Afterwards we have to fill in the details, and to check whether the plan really works.*"

These insights make proof planning intriguing for interactive as well as for automated theorem proving.
Bundy [2] and his group in Edinburgh pioneered *proof planning* as a technique that can be considered AI-planning and that employs an intelligent guidance of proofs. This work resulted in the proof planner CL^AM [22] that plans proofs by mathematical induction and that performs little average search.
Proof planning contrast with the more local heuristics which have previously been used for search control (in automated theorem proving). That is, instead of making separate decisions at each choice point of proving at the (low) level of logical inferences, based on local clues, proof planning has some sense of the overall direction of the proof.
The global search control is achieved by joining two roads, (1) the use of tactics and (2) meta-level control:

1. As opposed to traditional automated theorem that applies calculus-level inference rules, i.e. low level inferences, proof planning relies on tactics [10]. Tactics are procedures that produce a (not necessarily fixed) sequence of lower level inferences when executed, for instance a sequence of logical inferences at the calculus-level. Previously, tactics have already been employed in several interactive theorem provers, e.g. Nuprl [6].
   In order to enable a combination of tactical theorem proving with meta-level control, Bundy [2] introduced *methods* as (partial) specifications of tactics that specify in a meta-language the preconditions and effects of its application.[3] In Figure 1 the structure of CL^AM's methods is depicted. The methods serve as planning operators whose application yields the sequents from the output slot as subgoals. These preconditions contain control knowledge. They

```
name:          Prolog term
input:         sequent H==>G, H set of hypotheses, G goal
precondition:  list of conjuncts in meta-level language
postcondition: list of conjuncts in meta-level language
output:        list of sequents
tactic:        Prolog term
```

Figure 1. The method data structure in CL^AM.

   describe in a meta-language (1) syntactic properties of object-level expressions (sequents and formulas) E that are subgoals or elements of the initial state (definitions or axioms) or (2) abstract properties of these expressions that emerge from *annotations* to

[1] Universität des Saarlandes, Fachbereich Informatik, D-66041 Saarbrücken, Germany. email: melis@cs.uni-sb.de
[2] University of Edinburgh, Dept. of AI 80 South Bridge, Edinburgh EH1 1HN, UK. email: bundy@aisb.ed.ac.uk

[3] Note that these preconditions and effects of the methods in CL^AM denote things different from input and output which consist of sequents. Sequents are of the form $\Delta \vdash F$, where Δ is a set of formulas and F is a formula.

E. An example of the first kind of preconditions is "E has a free variable". An example of the second kind of preconditions is "There is a recorded definition R of the form (`lhs -> rhs`) such that the annotated `lhs` matches a subexpression of the annotated E while preserving the annotations."

2. The meta-level control came into play by (a) recognizing common proof plan patterns in families of proofs, for instance in proofs by mathematical induction or in diagonalization proofs, and by (b) discovering abstract goals and abstract heuristics that can guide the search for proofs.

   (a) Proofs by mathematical induction[4] reveal a common general structure displayed in Figure 2. This pattern is roughly to first

Induction
Base case
Symbolic evaluation
Simplification
Tautology checking
Step case
Rippling
Fertilization

Figure 2. Structure of proofs by mathematical induction

   find an appropriate induction schema and then to prove the conjecture for the base case, e.g., for $n = 0$, and for the step case, where the conjecture for a "successor" of the induction variable, e.g. of n, (called the induction conclusion) is proved provided the conjecture for the induction variable itself (which is called the induction hypothesis) holds. The step case pattern includes some kind of "fertilization", i.e., of applying the induction hypothesis to a rewritten induction conclusion such that a true formula results.
   The rewriting is subject to the abstract heuristic *rippling* described next.

   (b) A meta-level goal in the step case is to reduce the differences between induction conclusion and induction hypothesis in order to enable a final fertilization. These differences are represented by annotations, e.g., colours, to the induction conclusion. Axioms and definitions that belong to the initial state and which can be used to reduce the differences are annotated similarly.
   The abstract search heuristic for proofs by mathematical induction, *rippling*, was introduced by Bundy [2] and Hutter [12]. It describes a systematic way to remove the differences, for example by moving the differences outward until the induction hypothesis can be applied to an inner part of the rewritten induction conclusion. For example, in proving the conjecture

$$\forall x, y, z.\ x + (y + z) = (x + y) + z \tag{1}$$

   the induction hypothesis is

$$x + (y + z) = (x + y) + z \tag{2}$$

   and the conclusion is

$$\boxed{s(\underline{x})} + (y + z) = (\boxed{s(\underline{x})} + y) + z \tag{3}$$

   The boxes, excluding the underlined terms, denote the differences. The non-differences are called the *skeleton*. Rippling works by successively applying skeleton preserving definitions and axioms to the induction conclusion. A definition of the function + is

$$\boxed{s(\underline{U})} + V \Rightarrow \boxed{s(\underline{U + V})} \tag{4}$$

   where the skeleton on each side of the implication is $U + V$. In this example, rippling involves the repeated application of (4) which moves the differences outwards until the following expression is obtained

$$\boxed{s(\underline{x + (y + z)})} = \boxed{s(\underline{(x + y) + z})} \tag{5}$$

   to which the induction hypothesis can be applied.

   CL^AM's preconditions and effects of methods allow to plan (to reason about) the application of tactics not just by "precondition achievement" but supported by the meta-level control provided by rippling. In CL^AM a proof plan is then built as a tree of methods by searching the plan-space.

Due to the urgently needed search control in theorem proving, proof planning became more popular recently. Apart from the system CL^AM, other experiments explore different ways to realize proof planning. For instance, the proof planner of Omega [11] performs state-space search. As opposed to CL^AM, it prefers more declaratively represented methods the output of which is determined by the input. Those methods can be subject to reformulations. The method representation in Omega allows for different level of goals which naturally leads to hierarchical planning.

The fact that proof plans are an *abstract* and *structured* representation of proofs makes proof planning and, in particular, proof plans attractive for other activities in theorem proving:

- The abstract and structured representation is well-suited for theorem proving by analogy, as shown in [17]. The structure of a proof plan can be exploited when analogically transferring methods and subplans. As opposed to analogy at an abstract level, the analogical transfer often fails if drawn at the low level of logical inference rules.
- For derivational analogy [5, 24, 17], proof plans that store control information are needed.
- Proof plans that store control information are also well-suited for the explanation of proofs and for user interaction as, e.g., pursued in Barnacle [16], an interactive version of CL^AM. There, explanations are extracted from the preconditions of CL^AM's methods.
- As demonstrated in [15], a structured presentation of a proof has proved very important for the human understanding of proofs and is, therefore, needed for proof presentation and interactive theorem proving.

[4] Mathematical induction is a generalization of the well known pattern of Peano induction over natural numbers that has the induction schema $\frac{P(0), \forall k(P(k) \rightarrow P(k+1))}{\forall n(P(n))}$.

3 Properties of Proof Planning

Proof planning is a specific plan formation in the "precondition achievement" sense of AI-planning, and as an experiment the author has implemented a simple theorem proving domain in Prodigy. Specific characteristics of this "domain" are:

- The objects are (mathematical) objects such as numbers, lists, or trees and actions are manipulations of formulas describing objects and their relations and functions.
- Formal mathematical proofs which theorem proving aims at, are often long and very complex and even proof plans can be very deep. Therefore the need to represent bigger chunks by a planning operator and to understand a proof plan as an abstract representation of a proof.
- Goal interaction which is a major issue for plan formation in general and led to the development of partial order planning [21, 20]. In proof planning there is no goal interaction in the original object-level sense because the application of a sequence of logical inference rules does not destroy object-level preconditions.
- The acquisition of methods and of control knowledge for mathematical domains is difficult. It took, for instance, quite some time to polish CL^AM's methods that at first simulated procedures from the Nqthm theorem prover. Now CL^AM gets a long way with few methods for planning inductive proofs. We expect the acquisition of control knowledge and methods to be a major research problem for mathematical domains.
- In proof planning, the knowledge about the mathematical world is complete and certain rather than incomplete and uncertain as in many real world applications of planning.

4 Lessons from Planning and Proof Planning

As Weld [25] summarizes, in planning, knowledge-based search via a miniature production system turned out to be a good idea. Usually, these rules refer to local decisions. They can, however, also express control knowledge referring to the global development of a plan. First, in SOAR [14] such a control was explored and in the Prodigy system [18] the ideas were refined. Such a control is also described in [1]. Meta-level control-rules can be found in Press [4]. In Prodigy the control-rules contain meta-predicates that refer to the current state, the sequence of operators, etc. The experiences with a separate body of control-rules in Prodigy are summarized in [23]: The advantage of the factual-control knowledge distinction are modularity, reification of the control knowledge, selectivity in building learning modules, and compositionality of the acquired control knowledge. In SOAR and Prodigy matching algorithms and data structures have been developed to cope efficiently with many control-rules (see [7]). These experiences can help to design proof planning systems that make use of the advantages mentioned.

Currently, the most interesting feature of proof planning that could fertilize planning seems to be that abstract goals are pursued by heuristics expressed in a meta-language. In particular, the abstract control knowledge might be of interest for planning in real environments. The design of meta-predicates that capture proof-relevant abstractions, e.g., those involved in rippling, gives an additional means of control and thus, adds to the power of the proof planner. Thereby proof planning becomes more than pure precondition achievement.
Macro-operators as investigated in [13] are a first step towards plan patterns and therefore of interest for proof planning. They correspond to fixed patterns of (sub)plans. For proof planning, however, we need to find even more flexible patterns in order to structure a proof as the experience with proofs by mathematical induction shows. In CL^AM, an iteration over several submethods can provide a flexible pattern represented by a supermethod. This idea might help in planning to obtain plan patterns/macros that can be expanded flexibly.
Some clever hierarchical planning is needed in proof planning. So far, two different techniques for hierarchical planning are used. CL^AM realizes hierarchical planning by flexibly expanding (super)methods to a sequence of (sub)methods from a subset of methods. The output of a super-method is determined by applying the submethods. In Omega hierarchical planning is realized by explicitly marking less relevant proof obligations (subgoals) that can be planned for in the next lower hierarchical level. This will be amended by meta-level criteria for distingishing the hierarchy levels.

REFERENCES

[1] A. Barrett, K. Golden, J.S. Penberthy, and D. Weld, *USPOP User's Manual, Version 2.0*, Dept.of Computer Science and Engineering, University of Washington, 1993. Technical Report 93-09-06.

[2] A. Bundy, `The use of explicit plans to guide inductive proofs', in *Proc. 9th International Conference on Automated Deduction (CADE)*, eds., E. Lusk and R. Overbeek, volume 310 of *Lecture Notes in Computer Science*, pp. 111–120, Argonne, (1988). Springer.

[3] A. Bundy, `Proof planning', in *Proceedings of the International Conference on Planning 1996 (AIPS-96*, (1996).

[4] A. Bundy and B. Welham, `Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation', *Artificial Intelligence*, **16**(2), 189–212, (1981). Also available from Edinburgh as DAI Research Paper 121.

[5] J.G. Carbonell, `Derivational analogy: A theory of reconstructive problem solving and expertise acquisition', in *Machine Learning: An Artificial Intelligence Approach*, eds., R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, 371–392, Morgan Kaufmann Publ., Los Altos, (1986).

[6] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panagaden, J.T. Sasaki, and S.F. Smith, *Implementing Mathematics with the Nuprl Proof Development System*, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[7] R.B. Doorenbos, *Production Matching for Large Learning Systems*, Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, January 1995.

[8] M. Drummond, `On precondition achievement and the computational economics of automatic planning', in *Current Trends in AI Planning*, 6–13, IOS Press, (1994).

[9] G. Faltings and U. Deker, `Interview: Die Neugier, etwas ganz genau wissen zu wollen', *bild der wissenschaft*, (10), 169–182, (1983).

[10] M. Gordon, R. Milner, and C.P. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, Lecture Notes in Computer Science 78, Springer, Berlin, 1979.

[11] X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann, `Omega-MKRP: A Proof Development Environment', in *Proc. 12th International Conference on Automated Deduction (CADE)*, Nancy, (1994).

[12] D. Hutter, `Guiding inductive proofs', in *Proc. of 10th International Conference on Automated Deduction (CADE)*, ed., M.E. Stickel, volume Lecture Notes in Artificial Intelligence 449. Springer, (1990).

[13] R.E. Korf, `Macro-operators: A weak method for learning', *Artificial Intelligence*, **26**, 35–77, (1985).

[14] J Laird, A. Newell, and P. Rosenbloom, `SOAR:an architecture for general intelligence', *Artificial Intelligence*, **33**(1), 1–64, (1987).

[15] U. Leron, `Structuring mathematical proofs', *The American Mathematical Monthly*, **90**, 174–185, (1983).

[16] H. Lowe, A. Bundy, and D. McLean, `The use of proof planning for cooperative theorem proving', Research Paper 745, (1995). Submitted to

the special issue of the Journal of Symbolic Computation on graphical user interfaces and protocols.

[17] E. Melis, `A model of analogy-driven proof-plan construction', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 182–189, Montreal, (1995).

[18] S. Minton, C. Knoblock, D. Koukka, Y. Gil, R. Joseph, and J. Carbonell, *PRODIGY 2.0: The Manual and Tutorial*, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1989. CMU-CS-89-146.

[19] G. Polya, *How to Solve it*, 2nd ed. Doubleday, New York, 1957.

[20] E.D. Sacerdoti, `The nonlinear nature of plans', in *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pp. 206–214, (1975).

[21] A. Tate, `Generating project networks', in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 888–893. Morgan Kaufmann, (1977).

[22] F. van Harmelen, A. Ireland, S.Negrete, A. Stevens, and A. Smaill, `The CLAM proof planner, user manual and programmers manual', Technical Report version 2.0, University of Edinburgh, Edinburgh, (1993).

[23] Manuela Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe, `Integrating planning and learning: The PRODIGY architecture', *Journal of Experimental and Theoretical Artificial Intelligence*, 81–120, (1995).

[24] M.M. Veloso, *Planning and Learning by Analogical Reasoning*, Springer, Berlin, New York, 1994.

[25] D.S. Weld, `An introduction to least commitment planning', *AI magazine*, **15**(4), 27–61, (1994).