Erica Melis' Main Research Contributions

Already at Humboldt-University I have been interested in artificial intelligence (AI). Since 1992 I have been working on proof planning and on analogy in theorem proving. The main issues here are

1. automated *knowledge-based proof planning* and
2. *analogy-driven proof plan construction.*

1 The Setting

Automated theorem proving (ATP) is concerned with the mechanization of the deductive process. One of the long-term goals of ATP is the full mechanization of the proof capabilities of a mathematician. The formal verification of properties of programs, the synthesis of software, and a support for mathematical research and education belong to today's and tomorrow's applications of ATP and interactive theorem proving.

Traditional ATP is based on general-purpose machine-oriented logical calculi such as the resolution calculus, wherethe inference rules of such a calculus span the search space. In spite of many attempts, the inference steps defined by these calculi are rather small and low-level when compared with the proof steps of a trained mathematician and the search heuristics are rather local, syntactic, and extremely general-purpose. Because of the combinatorial explosion of the search space, traditional systems cannot prove difficult theorems in well-understood mathematical areas, although there are exceptions; for example, the mechanical proof of the open Robbins problem. Moreover, their output is hardly readable for an untrained user.

A different principle has been pursued by interactive theorem proving (see LCF [8], NuPrl [7], or Isabelle [34]), where tactic-based systems provide tactics that are realized by programs. Executing a tactic yields a *sequence* of calculus-level proof steps, i.e. a large step, and thus relieves the user from applying too many single inference rules in a row.

The overall motivation of the work in proof planning and theorem proving by analogy has been the disappointment with the results in traditional ATP that caused the ΩMEGA group as well as others to develop another vision for automated theorem proving.

Proof planning was originally conceived as a mere extension of tactical theorem proving [2]. The idea is as follows: Plan operators, called methods, are created from tactics by adding specifications: preconditions, postconditions, and application conditions. Proof planning searches for a plan, i.e., for a sequence of methods, where the precondition of a method matches the postcondition of its predecessor in that sequence.

2 Knowledge-Based Proof Planning

Problems The first proof planner, *CLAM*[3], was designed to prove theorems by mathematical induction. *CLAM* employs the rippling search heuristic for difference reduction [9, 4]. The difference reduction search heuristic works for equational proofs and proofs by mathematical induction. However, there are many theorems that are difficult or impossible to prove in *CLAM* systems, for instance, the limit theorem LIM* (the product of limits of two functions is equal to the limit of the product of these functions) because (i) for many problems, the means of control is insufficient and not flexible enough, (ii) no domain-specific methods are introduced, and (iii) the construction of mathematical objects with certain theory-specific properties is not supported.

Solutions (Summary) Prior to my work in proof planning, the ΩMEGA group in Saarbrücken changed their goals to proof planning and developed a representation of methods with a schematic expansion and a recursive expansion to calculus-level. However, guidance of the proof search, acquisition of common proof structures or frequently used inference procedures, instantiation of existentially quantified variables, and more abstract proof presentation were not investigated.

In order to make progress I introduced *knowledge-based* proof planning in which a general-purpose proof planner makes use of mathematical domain knowledge including methods, control-rules, and domain-specific external reasoners [18, 27]. This common principle of AI was not generally accepted in traditional theorem proving but has proven very fruitful when it comes to realistic maths. Furthermore, I introduced explicit and modular mathematical *search control* knowledge into theorem proving that allows for meta-level reasoning about a proof attempt. This idea has established a new point of view that has been recognized and adopted, e.g. in *λCLAM*. I also employed other techniques to restrict the search in proof planning such as *constraint solving*. Moreover, I developed a *proof presentation* at the level of proof plans that is well-suited for the communication with a user [23]. Although this idea seems obvious – it was original and not realized before. Last but not least, with a *case study* for the class of limit theorems I have illustrated that, indeed, knowledge-based proof planning solves some of the problems mentioned above. The knowledge acquisition for this case study required finding and representing frequently used methods and control knowledge and is a research result in itself [21]. As a *result*, several theorems from this class that could not be proved by any other (general-purpose) system can now be proved in ΩMEGA, e.g., LIM*.

The actual implementation of knowledge-based proof planning and proof presentation in ΩMEGA relies on the work of other members of the group and of two of my students.

2.1 Search Control

Automated meta-level reasoning in theorem proving is a feature that is unique to proof planning because in traditional ATP it is impossible and in tactical theorem proving the meta-reasoning is performed by the user.

Problems Although most methods encapsulate a chunk of calculus-level proofs, and therefore, proof plans are generally much shorter than the corresponding expansion into a calculus-level proof, the potential search spaces are still prohibitively large. The reason is that usually a large number of alternative methods is usually applicable at each choice point and, more seriously, a potentially infinitely branching search space cannot generally be avoided in mathematics (see [11]). The means of control should therefore be expressive, modifiable, and extendable and should be represented at a level that can easily be communicated to and designed by a user.

Solutions In mathematics, informed search is possible, if the existing control knowledge is explicated. Since proof planning methods represent mathematically meaningful steps, control knowledge can express mathematical 'intuition' rather than just syntactical information, such as an order or a number restriction on literals as in traditional ATP. My approach distinguishes 'local' control knowledge that determines conditions of the method's application, i.e., properties and relations that must hold for the sequents to be processed by the method, and 'global' control knowledge that refers to the planning state, the planning history (i.e., the failed proof attempts etc.), as well as time resources, the user model, and other global settings such as the typical examples of the theory within which the problem is stated. While legal, local knowledge is appropriately encoded into the application conditions of methods, the heuristic and global knowledge is now encoded into control-rules. Control-rules are formulated in a meta-language with decidable predicates.

In knowledge-based proof planning, at each decision point of the planner (decisions for a goal, a method, and an instantiation) a control unit evaluates the control-rules and guides the proof planner in its choice between several candidates, similar to an expert system, where the list of candidates rules is reduced to a smaller list. Control-rules have an antecedent and a consequent. The antecedent specifies the overall context within which this control-rule is applicable; its evaluation determines whether the control-rule is rejected or not.

For example, the following rule expresses the heuristic that if the method `Rewrite` whose parameter is instantiated by a rule `(C -> R)` is not applicable because the formula `C` is not (trivially) provable, then a `CaseSplit` method should be introduced into the plan.

```
(control-rule case-analysis-intro
       (IF  (last-method (Rewrite (?C -> ?R))) AND
                (failure-condition (trivial ?C)))
       (THEN (select (CaseSplit (?C or not ?C)))))
```

The control-rules are modularly represented. This is useful because the same method can be used in different theory contexts with different control. Consider, e.g., a method that translates a goal into a statement about natural numbers, i.e., a goal that can then be shown by Peano Induction. The function that determines the natural number is a parameter of a very general 'translation' method. For instance, in many completeness proofs for logic calculi this is a function that returns the excess literal number. However, also the length of clauses or the number of literal occurrences might be useful. In other theories the choice of this number may be different, e.g., the length of a derivation, the complexity of a formula, the number of grammar rule applications etc. Now, the translation method is very general and applicable in many mathematical domains, whereas the control knowledge that determines the parameter (the function) is specific and thus usually different in different domains.

Other well known reasons for using control-rules are: Modularly implemented and declaratively represented control-rules ease the *modification and extension* of the control knowledge. For instance, when new control information is required for new cases of problem solving, the existing control-rules can be modified or new control-rules can be introduced easily, as opposed to a re-implementation of the procedurally implemented control component of a planner. In particular, when the control is changed or *generalized* or when new methods are introduced into the domain, no re-implementation of the affected methods is necessary. Moreover, the declarative representation of control information by rules can be a basis for *automatically learning* control-rules, as realized in some planning systems, e.g., in [32, 1, 13].

Control-rules are new in proof planning and in automated theorem proving. In problem solving/planning, however, SOAR [12] and then Prodigy [33] were the first systems which used control-rules. My experience and discussions at CMU and Edinburgh were very helpful in order to find and bring about this meta-reasoning approach.

2.2 Integration of Constraint Solvers

In many mathematical proofs, logical steps are naturally combined with some form of specialized reasoning or effective computing. For instance, many proofs require the construction of an object with theory-specific properties. This is usually indicated by an existentially quantified variable in the theorem. More often than not, unification offers little support for this task and logic proofs, say of linear inequalities, can be very long and infeasible for purely logical automated theorem proving. However, theory-specific systems and algorithms can perform such services more efficiently because they use specialized data types and rely on efficient special-purpose algorithms.

Problems Hence, a problem is to *find witnesses* for existentially quantified variables. Even if we decide to use a domain-specific constraint solver, the *integration* of external systems into provers can be very difficult as known from ATP. Moreover, since off-the-shelf constraint solvers are designed to tackle other problems the following peculiarities in proof planning require *extensions* of constraint solvers: (i) the expansion of the proof plan should be logically correct; (ii) the constraints contain symbolic constants and 'user-defined' functions that do not belong to the signature of the original constraint domain but originate from the proof problem; (iii) not every variable to be constrained is known when the constraint solver is initialized.

Solutions Witnesses for existentially quantified variables are found by delaying the instantiation and by *integrating* an external constraint solver with proof planning that incrementally restricts the possible instantiation.

The integration scheme comprises (1) the method `Solve` whose application conditions may pass a constraint to the constraint solver via the functions `tell` and `ask`. When `tell` or `ask` respectively returns *true*, i.e., the constraint at hand is consistent with or entailed by the constraint store, then the method `Solve` is applicable. That is, the constraint solver restricts the search space of planning. (2) When no open goal is left in planning, an *answer constraint* C is computed by the constraint solver that expresses the collected constraints. C is then introduced into the (partial) proof plan. (3) From the collected (and propagated) constraints the constraint solver computes instantiations/values of variables which are instantiated all over the proof plan.

The integration of external reasoners as described in [19] takes care of the construction of mathematical objects. This *guides* and *simplifies* the formal proof [28]. In particular, a constraint solver may restrict the search space (guidance) and for witnesses synthesized by a constraint solver only the verification needs to be formally proved (simplification).

The extensions of constraint solving that are necessary for proof planning are essentially a planner-guided exploration of constraint states w annotated by their context, the introduction of additional constraints, e.g., for logical Eigenvariable conditions, an explicit computation of the answer constraint, and a synergetic symbolic and numeric constraint inference and search[25]. For realizing these extensions, the cooperation with Prof. Smolka's group was very helpful.

2.3 ***Knowledge-Based*** Proof Planning

Mathematicians have accumulated mathematical problem solving knowledge over hundreds of years and a mathematician is an expert in a highly specialized mathematical field rather than a universal expert. That is, domain-specific knowledge is a key for success while, of course, some general techniques and mathematical knowledge are important too.

Problem Most ATP systems use general, logical knowledge only and provide exactly the axioms needed for the proof with the problem.

Solution Knowledge-based proof planning can use domain-specific methods rather than general ones. For instance, a domain-specific decomposition trick that is *frequently used* for estimations in ϵ-δ-proofs is represented by the `ComplexEstimate` method that reduces the estimation $|b| < \epsilon$ of a complex term b to three less complex estimations $|k| \leq \mathbf{M}$, $|a\sigma| < \epsilon/(2 * \mathbf{M})$, and $|l| < \epsilon/2$ and proposes the existence of an appropriate real number $\mathbf{M}$ (auxiliary variable). The method is applicable, if b can be represented as $b = k * a\sigma + l$. Another example of a theory-specific method encodes the computation of integrals, whereas, e.g., `CaseSplit`, `Focus` setting a focus on a subformula and `UnwrapHyp` that unwraps a focused subformula of an assumption are theory-independent. The `Solve` method is not specific to one domain but can be used with any constraint domain.

2.4 Presentation of Proofs at the Plan-Level

One of the main problems in proof presentation is to provide an appropriate level of abstraction. Moreover, mathematicians and psychologists found empirically that, as opposed to the standard linear presentation, hierarchically structured proofs are more communicative of the mathematical ideas [14, 6].

Problems A traditional ATP system produces an output that is unreadable for mathematicians and does not support explanations. Previous proof presentation in natural language has recently been realized, e.g., in THINKER, ILF, and PROVERB. However, since they verbalize resolution proofs or only slightly abstracted proof representations, their presentations are not hierarchically structured, often not abstract enough, and cannot contain explanations.

Solutions The proof plan representation is the basis for automatically generating proof presentations satisfying the requirements to be more abstract and hierarchically structured [23]. This is a case, where a change of the underlying knowledge representation totally changes the problem solving horizon: Because of the switch from calculus-level proofs of traditional ATP to proof plans, problems become simple which were hardly solvable before. Proof plans are hierarchically structured and their methods can be (schematically) verbalized. Moreover, the expansion of methods allows to go into more detail if the user wants to and the explicit control knowledge provides information for building explanations on why something has been done in the proof. In ΩMEGA we have now a local presentation of methods which the user can access by clicking the method nodes in the graphical interface as well as a global presentation of the whole proof that is produced from the relevant verbalizations of methods in a proof plan.

3 Analogy in Theorem Proving

Analogy in automated theorem proving has received limited attention despite its importance in mathematics and is still a challenge. Early approaches, e.g. by Munyer, have been shown to be insufficient by Owen. Approaches based on second-order matching such as deLaTour and Caferra's, Kolbe and Walther's, and Curien's, have advanced the state of the art. They deal, however, with calculus-level proofs and with relatively restricted modifications of proofs only and therefore they provide a limited scope only.

Problems Computational accounts of theorem proving by analogy have been dominated by the idea of mapping symbols of the source theorem to symbols of the target theorem and employing an extended symbol map for transferring single proof steps of the source to the target. This proved to be inappropriate for many analogies drawn by mathematicians because in mathematical proofs often (abstract) proof ideas are transferred rather than just single calculus steps. For many analogies just symbol mapping is insufficient and a change of a problem representation, say, by unfolding a definition, may be necessary to reveal the commonality of theorems or assumptions, upon which the analogical transfer is based.

Solutions My analogy-driven proof plan construction [15] works at the level of proof *plans*. It is a derivational analogy ([5]) rather than a transformational analogy, i.e., it replays planning decisions from the source if their justifications hold. Hence, it is a control strategy in proof planning that employs a source proof plan to guide the construction of a proof plan for the target problem. This analogy has much in common with Veloso's general case-based planning [35]. Moreover, the approach includes a reformulation of the source proof plan. Moreover, I analyzed when it is reasonable to employ analogy [16]. Since analogy requires an overhead for retrieval and repair-matching, recently I suggested to employ analogy as one of the strategies in a multi-strategy proof planner [29] whose invocation can be controlled. This is in accordance with empirical finding on when humans use analogy in problem solving.

Results Analogy-driven proof plan construction has been implemented within the proof planners *CLAM* in Edinburgh [31] and ΩMEGA (in Saarbrücken). ABALONE, the system in Edinburgh, employs an abstraction implicit in rippling (labelled fragments [10]) to restrict the possible mappings [17]. Preserving this abstraction makes it likely that the proof will succeed in the target. Moreover, ABALONE distinguishes different occurrences of function symbols in the source such that they may have different images in the target. In *CLAM* also an internal analogy was realized [30], where internal analogy replays costly (repair) decisions in one and the same proof process.

The analogy in ΩMEGA ssucceeded in solving a challenge problem posed by Bledsoe: to prove a Heine-Borel theorem for the two-dimensional space by analogy to the proof of the Heine-Borel theorem for the one-dimensional space. Here the source proof was provided by Bledsoe and I collaborated with Woody Bledsoe on the solution of this problem. It turned out that with one of the reformulations I had used earlier the analogy could be realized [20].

An interesting application of my approach has been the reuse of interactively found proofs in formal software verification [26]. For this application, the derivational approach has been crucial because in tactical theorem proving (as well as in proof planning) the decisions of the proof process have to be replayed. Since the amount of user interaction is a prime cost factor in interactive program verification, the reuse of previous proofs can help reducing these costs considerably.

4 Prospectus

The two main application areas of my work I shall work in are education systems and formal software verification. Numerous theoretical and practical problems have still to be solved to make proof planning and analogy a powerful and reliable technology. My research becomes much broader now through the supervision of PhD students [22, 24] and the collaboration with other researchers.

For instance, currently I investigate proof planning and mathematical meta-reasoning for new mathematical domains in collaboration with PhD students Martin Pollet and

Volker Sorge (Saarbrücken). This research is being extended to a formal collaboration with CMU (Wilfried Sieg, Dana Scott, and Herb Simon), Eindhoven (Arjeh Cohen), and Cornell (Bob Constable). Because of the knowledge acquisition bottleneck, machine learning is one of the topics that I shall investigate in collaboration with Carla Gomes and Bart Selman (Cornell), Manuela Veloso (CMU), Daniel Borrajo (U. Madrid), and Manfred Kerber (U. Birmingham) as well as the PhD student Andreas Meier (Saarbrücken). I collaborate on learning structure and control knowledge from maths textbooks with Michael Kohlhase. For education systems a multimodal adaptive presentation with explanations and lemma construction will be necessary. On this topic I shall collaborate with Cornell and the PhD student Armin Fiedler. I believe that this research will be an important step towards computer-aided and automated theorem proving in realistic settings.

References

[1] D. Borrajo and M. Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 10:1–34, 1996.

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

[3] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[4] A. Bundy, F. van Harmelen, A. Ireland, and A. Smaill. Extensions to the rippling-out tactic for guiding inductive proofs. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.

[5] J.G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. Morgan Kaufmann Publ., Los Altos, 1986.

[6] R. Catrambone. Generalizing solution procedures learned from examples. *Journal of Experimental Psychology, Learning, Memory, and Cognition*, 22(4):1020–1031, 1996.

[7] 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, 1986.

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

[9] D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *Proceedings of 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.

[10] D. Hutter. Synthesis of induction orderings for existence proofs. In A. Bundy, editor, *Proceedings of 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 29–41. Springer, 1994.

[11] G. Kreisel. Mathematical logic. In T. Saaty, editor, *Lectures on Modern Mathematics*, volume 3, pages 95–195. J. Wiley & Sons, 1965.

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

[13] C. Leckie and I. Zukerman. Inductive learning of search control rules for planning. *Artificial Intelligence*, 101(1-2):63–98, 1998.

[14] U. Leron. Structuring mathematical proofs. *The American Mathematical Monthly*, 90:174–185, 1983.

[15] E. Melis. A model of analogy-driven proof-plan construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 182–189, Montreal, 1995.

[16] E. Melis. When to prove theorems by analogy? In *KI-96: Advances in Artificial Intelligence. 20th Annual German Conference on Artificial Intelligence*, volume 1137 of *Lecture Notes in Arrtificial Intelligence*, pages 259–271. Springer, 1996.

[17] E. Melis. Solution-relevant abstractions constrain retrieval and adaptation. In D. Leake and E. Plaza, editors, *Case-Based Reasoning Research and Development. Proceedings of the Second International Conference on Case-Based Reasoning*, volume 1266 of *Lecture Notes on Artificial Intelligence*, pages 383–392. Springer, 1997.

[18] E. Melis. AI-techniques in proof planning. In *European Conference on Artificial Intelligence*, pages 494–498, Brighton, 1998. Kluwer.

[19] E. Melis. Combining proof planning with constraint solving. In *Proceedings of Calculemus and Types'98*, 1998. Electronic Proceedings http://www.win.tue.nl/math/dw/pp/calc/proceedings.html.

[20] E. Melis. The Heine-Borel challenge problem: In honor of Woody Bledsoe. *Journal of Automated Reasoning*, 20(3):255–282, 1998.

[21] E. Melis. The "limit" domain. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence in Planning Systems*, pages 199–206, 1998.

[22] E. Melis. Proof planning with multiple strategies. In B. Gramlich and F. Pfenning, editors, *CADE-15 workshop on Strategies in Automated Deduction*, pages 57–68, 1998.

[23] E. Melis and U. Leron. A proof presentation suitable for teaching proofs. In S.P. Lajoie and M. Vivet, editors, *9th International Conference on Artificial Intelligence in Education*, pages 483–490, Le Mans, 1999. IOS Press.

[24] E. Melis and A. Meier. Proof planning with multiple strategies ii. In H. Kirchner B. Gramlich and F. Pfenning, editors, *FLoC'99 workshop on Strategies in Automated Deduction*, pages 61–72, 1999.

[25] E. Melis, T. Müller, and J. Zimmer. Extensions of constraint solving for proof planning. Seki report SR-99-04, Universität des Saarlandes, FB Informatik, April 1999.

[26] E. Melis and A. Schairer. Similarities and reuse of proofs in formal software verification. In B. Smyth and P. Cunningham, editors, *Advances in Case-Based Reasoning*, volume 1488 of *Lecture Notes in Artificial Intelligence*, pages 76–87. Springer, 1998.

[27] E. Melis and J.H. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, November 1999.

[28] E. Melis and V. Sorge. Employing external reasoners in proof planning. In A. Armando and T. Jebelean, editors, *Calculemus'99*, pages 123–134, 1999.

[29] E. Melis and C. Ullrich. Flexibly interleaving processes. In K.-D. Althoff and R. Bergmann, editors, *International Conference on Case-Based Reasoning*, volume 1650 of *Lecture Notes in Artificial Intelligence*, pages 263–275. Springer, 1999.

[30] E. Melis and J. Whittle. Internal analogy in inductive theorem proving. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 92–105. Springer, 1996.

[31] E. Melis and J. Whittle. Analogy in inductive theorem proving. *Journal of Automated Reasoning*, 22(2):117–147, 1999.

[32] S. Minton. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[33] 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.

[34] L.C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–368. Academic Press, 1990.

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