

# IsaPlanner: A Prototype Proof Planner in Isabelle

Lucas Dixon and Jacques Fleuriot \*

School of Informatics, University of Edinburgh,  
{lucas.dixon, jacques.fleuriot}@ed.ac.uk

**Abstract.** ISAPLANNER is a generic framework for proof planning in the interactive theorem prover Isabelle. It facilitates the encoding of reasoning techniques, which can be used to conjecture and prove theorems automatically. This paper introduces our approach to proof planning, gives an overview of ISAPLANNER, and presents one simple yet effective reasoning technique.

## 1 Introduction

Proof planning [3] provides a framework for encoding and applying common patterns of reasoning. It has been successfully used in a number of first and higher order domains including mathematical induction [3], hardware verification [4], higher order program synthesis [7], and more recently nonstandard analysis [8]. Proof planning generates an abstract description of a proof, which is known as a *proof plan*. This is typically a tactic tree or compound tactic which can be executed in a theorem prover to derive a fully formal proof. Currently there are two main approaches to proof planning, which are embodied by the  $\lambda$ Clam proof planner [12], and the Omega system [9].

Isabelle [11] is a generic theorem prover typically used in an interactive fashion. It supports formal reasoning in a number of object logics, including Zermelo-Fraenkel set theory (ZF), constructive type theory (CTT), and higher order logic (HOL). Object logics are formed and manipulated by Isabelle's intuitionistic higher-order meta logic, which supports polymorphic typing. Its features include a searchable theorem database, generic automatic proof tools [10], and within HOL a mechanism for the automatic derivation of induction schemes [13].

We present ISAPLANNER, a generic framework in Isabelle that interleaves proof planning with the proof plan's execution. This is a novel approach which allows proof planning to take advantage of the powerful tactics available in Isabelle. We describe an example of encoding a simple reasoning technique which makes use of Isabelle's simplifier both as a proof tool and as an aid to conjecturing lemmas. The main aim of our framework is to allow mathematical and proof assistant specific techniques to be encoded and applied to problems. We hope that our approach will improve both the interaction with, and automation in, interactive proof systems.

---

\* This research was funded by EPSRC grant GR/N37414/01. The authors would also like to thank the anonymous referees for their constructive and helpful feedback.

## 2 Our Approach to Proof Planning

Our approach breaks proof planning into a series of *reasoning states* which capture ‘snapshots’ of the planning process. These are intended to reflect the expressible points in a user’s proof process. Each reasoning state contains: the current partial proof plan, a reasoning technique to be applied, and contextual information.

The partial proof plan is an executable tactic tree and whenever it is modified data such as the current subgoals and assumptions are updated accordingly. This allows the result of a tactic application to inform the proof planning process. For example, simplification often results in a subgoal from which an obvious lemma can be speculated.

Reasoning techniques, which can be generic or theory-specific, are encoded as functions from a reasoning state to a sequence of reasoning states. Each state in the sequence represents a possible way in which the technique can be applied. For example, a technique performing a proof by induction might result in a reasoning state for induction on each variable of an inductive type in the goal.

The contextual information captures any knowledge that might be applicable to the current proof process. The information available is defined by the theory being worked in and can be built up during proof planning.

Our approach is concerned with providing a framework for the construction of proof plans *and* formal proofs whereas tactic-based theorem proving, in contrast, concentrates on the derivation of calculus level proofs. Proof planning is performed by searching through the possible sequences of reasoning steps. This terminates when a desired reasoning state is found, resulting in a proof plan and a proved theorem, or when the search space is exhausted.

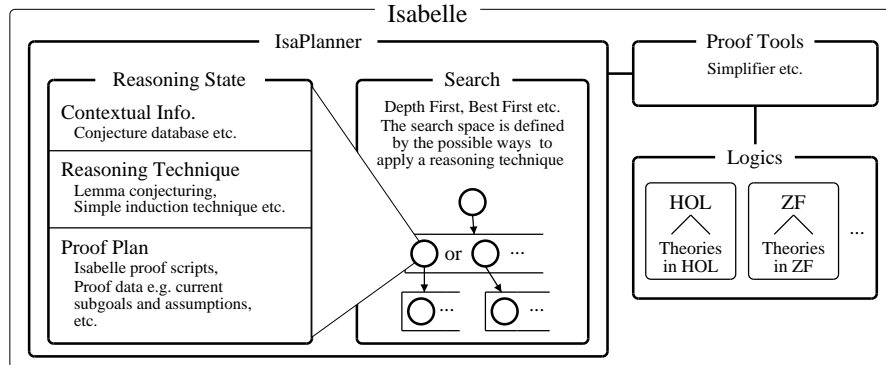
## 3 An Overview of IsaPlanner

ISAPLANNER is written in Standard ML and makes extensive use of Isabelle’s generic tools such as lazy sequences and discrimination nets. We present an overview of the system in Figure 1. At any one point in the proof planning process, the central component is the reasoning state. This is defined as a datatype containing a proof plan, a reasoning technique, and an extendable table of contextual information.

We provide a collection of functions which modify and express proof plans as Isabelle proof scripts. For example, the script generated when proof planning the commutativity of addition is:

```
theorem "x + y = y + x"
  by (induct "x", simp add: lemma1, simp add: lemma2)
```

where `lemma1`:  $x+0 = x$  and `lemma2`:  $Suc(x+y) = x+Suc(y)$  are automatically conjectured and proved. These can then be renamed, if need be, and used for other proofs in Isabelle. We avoid re-processing unchanged parts of the proof plan by storing the intermediate subgoals and assumptions resulting from its execution.



**Fig. 1.** An overview of ISAPLANNER that shows the contents of a reasoning state.

The contextual information, which can be added to dynamically, facilitates the design and definition of reasoning techniques. Examples of such information include a conjecture database and a structure to describe how a proof plan was found.

A number of search mechanisms have been implemented in ISAPLANNER, including Depth First, Iterative Deepening, Breadth First and Best First. An interactive mode also exists which presents the user with a textual description of the reasoning states that result from applying a reasoning technique. It also gives a textual representation of the proof plan, and a description of the proof process at varying levels of detail. The user can then select which reasoning state to explore next, or to backtrack up the search tree.

ISAPLANNER has a language that provides a number of functions to make the encoding of reasoning techniques easier. In particular we provide functions to add tactics to the proof plan, conjecture lemmas, and combine existing techniques. These facilitate the development of critics [6] which can be used to analyse and suggest patches to failed proof attempts.

## 4 A Simple Induction Technique Encoded in IsaPlanner

A simple but effective technique employed by users doing inductive proof in Isabelle involves induction on a suitable variable followed by exhaustive rewriting. We have embodied this approach in a simple reasoning technique that combines Isabelle's induction and simplification tactics with a lemma speculation critic. This can be expressed in the reasoning technique language as:

```
simp_and_conjecture OR (induct_on_a_var THEN simp_and_conjecture)
```

Conjectures are speculated from examining the subgoals that simplification fails to solve. To avoid over-specialised conjectures, we use a simple notion of generalisation [2] that replaces re-occurring sub-terms with new variables.

For example, when proving  $x^{(y+z)} = x^y \cdot x^z$ , after induction has been applied to  $y$ , simplification reduces the step case to  $x \cdot (x^y \cdot x^z) = (x \cdot x^y) \cdot x^z$ . From this

subgoal our reasoning technique is able to conjecture the required lemma: the associativity of multiplication.

As a remark, we note that initially this technique was slow due to repeated conjecturing and proof planning of the same lemmas. We solved this problem by implementing a memoization technique that stores the result of previously proof planned conjectures in a database.

## 5 Some Results from using the Induction Technique

This simple reasoning technique has been applied to 400 problems in the domains of ordinal and Peano arithmetics and list theory. Proof planning terminated on all examples and was able to conjecture the needed lemmas and make appropriate generalisations. This significantly improves on the automation currently provided by Isabelle. Figure 2 shows a table highlighting some of the results.

Domain	Theorem	Time (in seconds)	Lemmas Proved
<i>Properties of Lists</i>	$rev(map\ f\ xs) = map\ f\ rev(xs)$	0.4	1
	$rev(xs\ @\ ys) = rev(ys)\ @\ rev(xs)$	0.4	2
<i>Peano Arithmetic</i>	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	0.8	3
	$x^{(y+z)} = (x^y) \cdot (x^z)$	1.0	8
	$x \cdot (y \cdot z) = y \cdot (x \cdot z)$	1.4	9
<i>Ordinal Arithmetic</i>	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	0.1	6
	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	0.3	8
	$(x^y)^z = x^{(y \cdot z)}$	0.9	16

**Fig. 2.** Some results using ISAPLANNER to prove theorems in ordinal arithmetic, Peano arithmetic, and list theory, that show the time and number of lemmas conjectured and proved automatically. The timings were obtained from a 1GHz Intel PC with 256MB of RAM, and using Isabelle2002 with PolyML.

In an earlier version of ISAPLANNER, we also implemented a higher order version of rippling based on the work of Dennis and Smaill in  $\lambda$ Clam [5]. However, we found that the simple induction technique presented in this paper was much quicker than our implementation of rippling. We also found that our induction technique was also significantly quicker than rippling in the  $\lambda$ Clam system. For example, proof planning the theorem  $x^{(y \cdot z)} = (x^y)^z$  in ordinal arithmetic, took over 5 minutes in  $\lambda$ Clam, compared with 0.9 seconds in ISAPLANNER. One notable example which our technique cannot solve is the theorem  $rev(rev(l)) = l$ , in the theory of lists. Rippling in the  $\lambda$ Clam system is able to prove this by making use of a more sophisticated lemma speculation and generalisation critic [6] which we are in the process of implementing.

## 6 Conclusion and Further Work

We have presented ISAPLANNER, a generic framework for proof planning in Isabelle, and described a simple but effective reasoning technique users employ in interactive proof. This technique has been encoded in the system and used to generate Isabelle proof plans for problems in a variety of domains, with impressive speed. Another salient aspect of our approach is the derivation of fully formal proofs as part of the proof planning process. This contrasts with the approach taken in Clam and  $\lambda$ Clam for example, where one has to attach an object level theorem prover explicitly to the system in order to execute and validate the generated proof plan (see the work of Boulton et al. [1], for example).

Further work includes: extending the reasoning technique language, encoding and evaluating new reasoning techniques, and improving the user interface. In particular we plan to incorporate rippling into the latest version, as well as port some of the proof critics from the  $\lambda$ Clam system. We also intend to provide a readable version of the output proof plans using the more declarative Isar [14] language.

## References

1. R. Boulton, K. Slind, A. Bundy, and M. Gordon. An interface between CLAM and HOL. In *TPHOLs'98*, volume 1479 of *LNAI*, pages 87–104, 1998.
2. R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979. ACM monograph series.
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. Francisco Cantu, Alan Bundy, Alan Smaill, and David Basin. Experiments in automating hardware verification using inductive proof planning. In *FMCAD96*, volume 1166 of *LNCS*, pages 94–108, 1996.
5. L. A. Dennis and A. Smaill. Ordinal arithmetic: A case study for rippling in a higher order domain. In *TPHOLs'01*, volume 2152 of *LNCS*, pages 185–200, 2001.
6. A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.
7. D. Lacey, J. D. C. Richardson, and A. Smaill. Logic program synthesis in a higher order setting. In *Computational Logic*, volume 1861 of *LNCS*, pages 87–100, 2000.
8. Ewen Maclean, Jacques Fleuriot, and Alan Smaill. Proof-planning non-standard analysis. In *The 7th International Symposium on AI and Mathematics*, 2002.
9. E. Mellis and J. H. Siekmann. Knowledge-based proof planning. *Journal of AI*, 115(1):65–105, 1999.
10. L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3), 1999.
11. L.C. Paulson. *Isabelle: A generic theorem prover*. Springer-Verlag, 1994.
12. J. D. C Richardson, A. Smaill, and I. Green. System description: proof planning in higher-order logic with Lambda-Clam. In *CADE'98*, volume 1421 of *LNCS*, pages 129–133, 1998.
13. K. Slind. Derivation and use of induction schemes in higher-order logic. In *TPHOLs'97*, volume 1275 of *LNCS*, pages 275–290, 1997.
14. Markus Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *TPHOLs'99*, volume 1690 of *LNCS*, pages 167–184, 1999.