IsaPlanner 2: A Proof Planner for Isabelle

Lucas Dixon and Moa Johansson

School of Informatics, University of Edinburgh

Abstract. We describe version 2 of IsaPlanner, a proof planner for the Isabelle proof assistant and present the central design decisions and their motivations. The major advances are the support for a declarative presentation of the proof plans, reasoning with meta-variables to support middle-out reasoning, new proof critics for lemma speculation and case analysis, the ability to mix search strategies, and the inclusion of a higher-order version of rippling that can use best-first search. The result is a more flexible and powerful proof planner for exploring proof automation in Isabelle.

1 Introduction

Proof assistants, such as Isabelle [10], Coq [11] and HOL [7], provide a framework for formalisation tasks such software verification and mechanised mathematics. Typically, automation is developed by writing programs, called *tactics*, that combine operations from a small trusted kernel. Although many forms of proof automation are already available, developing new tactics and extending existing ones can be difficult. Higher-level concepts, such as search space and heuristic guidance, must be developed on top of the the logical kernel.

Proof Planning provides this kind of high-level machinery for encoding and applying common patterns of reasoning [2]. When encoded in a proof planner we call these patterns *reasoning techniques*. The first version of IsaPlanner, a proof planner for Isabelle, provided an efficient inductive prover [5]. However, it used a restrictive representation of proof plans that did not allow meta-variables in the goals. This limited experimentation with techniques based on middle-out reasoning, as proposed in [1], as well as with *proof critics* [8].

We present version 2 of IsaPlanner and motive the design decisions. The central change is a revised representation of proof plans which has lead to new ways of writing reasoning techniques and lifts the meta-variable restrictions. This provides the necessary machinery to experiment with middle-out reasoning and proof critics. We discuss the role this has in facilitating the expression of reasoning techniques and outline the lemma speculation and case analysis critics that have been implemented. The representation of proof plans and meta-variables also impacts on proof search. The new technique language allows search strategies to be locally specified and mixed, includes operations, such as filtering, on the search space, and extends the machinery for interactively tracing a proof planning attempt. These advances make IsaPlanner a more flexible tool for experimentation with proof planning and have lead to improvements in the power of the inductive prover.

2 Representation

Proof planning involves applying reasoning techniques to produce proof plans. Reasoning techniques differ from tactics in that they operate on hierarchical and partial tactic trees, called *proof plans*, rather than on the subgoals of a proof state. This allows them to affect earlier parts of the proof. For example, failure to prove the step case of an inductive proof can be used to suggest a change to the induction scheme [8].

The features that differentiate proof plans from other proof-representations are that they include explicit gaps, they allow steps to be hierarchical in the sense that they can contain a sub- proof plans, and they store information about how proof plans are found. IsaPlanner's representation also annotates gaps with *continuations* proposing how they might be filled. These features make proof plans introspectable and incrementally unfoldable.

The foundational changes to version 2 of IsaPlanner are in the representation of proof plans. In the previous version, a proof plan was a list of steps analogous to a procedural proof script. Each step holds Isabelle's proof state which contains the list of subgoals. Within each of these subgoals the assumptions are also stored in a list. In tactic languages and in the first version of IsaPlanner, goals and assumptions are referred to by specifying a location in the respective list. However, tactics often modify the ordering of these lists, for instance when they introduce new subgoals and assumptions. This makes referring to assumptions and goals unstable: after applying a tactic the location of a goal or assumption may change.

Version 2 of IsaPlanner provides a representation of proof plans, inspired by declarative proof languages such as Isar [12], that gives stable references to assumptions, goals and meta-variables in the form unique names. These proof plans do not step outside Isabelle's trusted kernel in the sense that an Isabelle theorem corresponding to the proof attempt can be generated. Contexts which are made up of the collection of assumptions and constants under which a goal is being proved, are also named uniquely. By discharging assumptions using these names we simplify as well as speed up the process of their discharge.

In contrast, Isabelle holds the assumptions to be discharged in a set which is made efficient by term-indexing. However, this representation does not allow meta-variables as it would require re-indexing after instantiation. IsaPlanner's use of explicit names overcomes this limitation because the names of assumptions are not affected by instantiation. The effect is that conjectures containing meta-variables can be used in a proof attempt, become incrementally instantiated, and their proof can be interleaved with other conjectures and subgoals, possibly sharing meta-variables. An example application for this kind of reasoning has been described in [1].

Internally, IsaPlanner stores meta-variables in a table which is indexed by their unique name and which holds the names of goals, assumptions and other meta-variables in which they occur. This allows the proof plan to be updated efficiently whenever a variable is instantiated. We also keep a record of the instantiations in order to support deductive synthesis. Using a purely functional

representation with explicit names rather than references allows efficient sharing of memory while maintaining the ability to search over different proof plans without having to do any explicit memory management.

3 The Technique Language

The purpose of the technique language is to facilitate developing and experimenting with new reasoning techniques for automation.

Reasoning techniques are writen as ML functions from a proof planning state, representing a point in the search space, to a list of possible next states which represent the or-choices from that point. Each state contains the *continuation* reasoning technique that will be applied next. Unlike tactics which hide search within their application, this allows reasoning techniques to express operations on the search space. For example, the technique constructor `MAP` takes a function on reasoning states f and a technique r and results in a new reasoning technique that behaves like r but has f applied to each intermediate state in the search space. For instance, this can be used to write a generic pruning function that filters out every state that matches a given criteria.

We can express techniques that affect other branches in the search space by using shared variables. Each branch has access to this variable and can affect it. This is used, for example, when a lemma is proved in one branch of the search space to eagerly complete other proof attempts of the same lemma that are in progress. Similarly, we can express a pruning technique that, when a conjecture that is proved false in one branch of the search space, stops all proof attempts in other branches.

The addition of explicit names for assumptions and subgoals provides language constructs that give better control over and-choices. They allow techniques to easily switch focus between the proof-attempts of different subgoals. Such switching of focus is particularly useful for middle-out reasoning where a step in one branch of the proof may instantiate a meta-variable that then makes another branch easier and thus suggests a change in focus. This happens when proving customised induction rules in deductive synthesis [1].

3.1 Tracing

In order to support debugging of complex techniques, as well as to help the development of new ones, IsaPlanner continues to provide an interface for interactively tracing through the search space [3]. This gives a text interface where the user can select which branch to explore next. In version 2, support has been added for stepping over sections of the search space as well as asking the prover to automatically search until some particular state is reached. This is useful for debugging as well as development.

3.2 Mixing Search Strategies

In the previous version of IsaPlanner, several search strategies were supported; however only one search strategy could be used per proof attempt. Although, like tactic languages, a second proof attempt with a different search strategy can be hidden within a single step, this would not allow the tracing machinery or the search space operations to see the hidden search space.

To allow a reasoning technique to specify a local search strategy and keep the advantages of an explicit representation of the search space, we developed a meta-search strategy. This examines each proof planning state for search operators that push or pop search strategies a stack. This allows us to provide a technique constructor `SEARCH` which takes a search strategy s and a technique r and informs the meta-strategy to pop s onto the strategy stack then apply r and finally pop s off the strategy stack. This allows search strategies to be specified by techniques and mixed within a single proof planning attempt. As mentioned below, this has been used to improve IsaPlanner's inductive proof technique [9].

4 Reasoning Techniques

IsaPlanner provides a higher-order dynamic version of rippling, a powerful rewriting technique that guides a proof by removing the syntactic differences between the goal and a specified term, usually an assumption [6]. In version 2, this has been designed in a modular fashion and used to implement and experiment with many variants of rippling [4].

The inductive prover implemented in IsaPlanner makes use of the rippling machinery for the step cases of inductive proofs and uses Isabelle's simplifier for the base cases. Recently, we experimented with mixing best-first search for rippling, and depth-first search over the rest of the proof attempt, to develop a new version of the inductive prover. This was shown to improve the technique, allowing it automatically prove theorems that were previously outside the search space [9].

4.1 Lemma Speculation Critic

In addition to the *lemma calculation* critic, used by the inductive prover and described in [4], IsaPlanner now also supports a more sophisticated *lemma speculation* critic, inspired by [8]. The lemma calculation critic uses common-subterm generalisation of the goal propose a lemma. In contrast, the lemma speculation critic creates a schematic lemma by inserting meta-variables standing for yet unknown term-structure. This schematic lemma is then applied to the goal and rippling-based rewriting continues. As meta-variables are shared across the goals in the proof plan, instantiations in the goal will also instantiate the lemma. This gives more flexibility to the lemmas being conjectured.

4.2 Case Analysis

Program specifications will frequently contain if- and case-statements. In order to automate such verification proofs, a case-analysis critic for if-statements has recently been developed. During rippling, the critic automatically introduces a case-split on the condition of the if-statement when it cannot automatically be reduced to true or false. This creates a subgoal for each branch of the if statement on which rippling can then continue. A similar critic for automating splitting of case-statements is under development.

5 Conclusions and Further Work

IsaPlanner now provides a representation of proof plans with appropriate machinery for handling meta-variables. It also allows the proof attempts of different subgoals to be interleaved. The purpose of the system is to support research into novel proof planning techniques and to allow experimentation with proposed ones such as middle-out reasoning and proof critics. A lemma speculation critic as well as a case analysis one have been already been implemented. We plan to further develop IsaPlanner's support for proof critics and deductive synthesis. Further work also includes providing a logical account of the new representation for proof plans.

References

1. A. Bundy, L. Dixon, J. Gow, and J. Fleuriot. Constructing induction rules for deductive synthesis proofs. *ENTCS*, 153(1):3–21, 2006.
2. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *JAR*, 7(3):303–324, 1991.
3. L. Dixon. Interactive hierarchical tracing of techniques in IsaPlanner. In *User Interfaces for Theroem Provers (UITP'05)*, ENTCS, 2005.
4. L. Dixon. *A Proof Planning Framework for Isabelle.* PhD thesis, University of Edinburgh, 2005.
5. L. Dixon and J. D. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In *CADE-03*, volume 2741 of *LNCS*, pages 279–283, 2003.
6. L. Dixon and J. D. Fleuriot. Higher order rippling in IsaPlanner. In *TPHOLs-04*, volume 3223 of *LNCS*, pages 83–98, 2004.
7. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic.* Cambridge University Press, 1993.
8. A. Ireland and A. Bundy. Productive use of failure in inductive proof. *JAR*, 16(1–2):79–111, 1996.
9. M. Johansson, A. Bundy, and L. Dixon. Best-first rippling. In *Reasoning, Action and Interaction in AI Theories and Systems*, volume 4155 of *LNCS*, pages 83–100, 2006. An extended version is available as University of Edinburgh Technical Report EDI-INF-RR-0849, http://www.inf.ed.ac.uk/.
10. L.C. Paulson. *Isabelle: A generic theorem prover.* Springer-Verlag, 1994.
11. The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.0*, April 2004.
12. M. Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *TPHOLs-99*, volume 1690 of *LNCS*, pages 167–184, 1999.