Structured Proofs from a Graphical Proof Strategy Language

Colin Farquhar\(^1\) Gudmund Grov\(^1\)
Heriot-Watt University, Edinburgh, Email: \{cif30,G.Grov\}@hw.ac.uk

**Abstract:** Previously, a graph based proof strategy language has been developed for writing high-level strategy, and a prototype implementation has been provided in the Isabelle theorem prover. Whilst the language can include features such as loops, one cannot view a generated proof from the strategy in a natural way, nor can one replay a proof without running the full strategy. Here, we discuss the underlying proof representation used by strategy language, and show how to export a generated proof into an Isabelle/Isar structured proof. This enables both replay and enables a user familiar with Isabelle to view a generated proof.

1 A Graphical Proof Strategy Language

In previous work [1], a graph based proof strategy language was introduced. Here, tactics are nodes in the graph, and edges are drawn between graphs to direct the goals. The graph is evaluated by sending the goals down the wire. A tactic is then applied to it, which generates sub-goals, and these are sent to the output edges. To direct the goals to the correct tactic (i.e down the correct edge), a notion of goal types was introduced, where a goal type gives a description of a goal. To illustrate, consider the strategy graph shown left-most in Figure 1. Here, there are two goal types: GT\(^1\) captures goals where the well-known intro tactic is applicable, while GT\(^2\) captures cases where it is not. Thus, only goals where the intro tactic is applicable are sent there.

The goal type enables a type safe way of composing, e.g. the THEN combinator can be encoded by plugging inputs of one graph to the outputs of another, ensuring that only edges of the same goal types are plugged together. Moreover, as shown in the strategy in Figure 1, repetition can be represented directly as a feedback loop.

Hierarchies are supported in the language, by nesting graphs inside a vertex. In fact, both tactics shown in the figure are nested: intro can apply a range of different introduction rules (e.g. for \(\&\), \(\imp\)), while Discharge eliminates conjunctions in the hypothesis and applies the assumption tactic.

2 Proof Representation

A prototype of the strategy language has been implemented on top of the Isabelle theorem prover [2]. In order to work with Isabelle tactics, and ensure soundness by running the proof through the Isabelle kernel, a proof representation which Isabelle understands is required.

Isabelle handles a goal \(G\) internally by creating a goal \(G \Rightarrow G\). A resolution step applied to \(G\) creates two new goals, for example \(F\) and \(H\), and updates the goal to \(F \Rightarrow H \Rightarrow G\). \(F\) and \(H\) are subgoals which have to be proven. The proof is completed when all subgoals have been discharged, i.e the only remaining goal is \(G\). Compared to traditional LCF, this means that in each step we are working with a safe \(\text{thm}\) type, and we don’t have to apply a justification function afterwards. However, for our usage this representation has three major drawbacks

- There is only goal, and each subgoal is just a hypothesis in this goal.
- Assumption are not named, meaning we cannot refer directly to a particular assumption. This is a problem for large conjectures.
- The proof history is not stored, which is a requirement to generate the Isar scripts discussed in the next section.

In a strategy graph, each sub-goal is a separate node in the graph. One could overcome this by simply keeping the index number of the hypothesis of the overall goal, however this does not solve the other two drawbacks.

Instead we have generated a proof representation which stores the full proof tree, and thus the full proof history. Such a tree is illustrated in the middle of Figure 1. This is the proof of \(A \land B \Rightarrow B \land A\) – as a result of applying the strategy on the right. Note that the outer-most boxes are to illustrate how the tree was generated. Currently, the proof representation does not reflect hierarchies from the strategies, and adding this is a matter for further work.

Each node in the tree represents a different step in the proof. It keeps track of: the goal to be proven; the subgoals generated; the nodes which prove the subgoals; and any assumptions and fixed variables. Note that it utilises Isabelle’s local context facilities to enable naming each assumption and keep track of fixed variables.

3 Creating Structured Proofs

The proof/strategy representations suffer from two drawbacks:

- The proof cannot be replayed.
- The proof cannot be viewed in a notation understandable for most users.

To overcome both these limitations we here develop a translation from the internal proof representation into an Isabelle/Isar structured proof [3]. The right-most side of Figure 1 shows the structured proof generated from the proof representation.
The Isar proof structure gives a step-by-step representation of our proof, displaying the proof of each subgoal as distinct “block” within the proof. These blocks are nested within each other, with the proof of the first subgoal containing the proofs of its own subgoals. This format has two main advantages as a proof representation: it can be interpreted by both the user and the theorem prover, and it gives more information about our proof than the graphical representation.

We can now describe the procedure for translating our proof representation into a structured script. We start with the root of the tree and project the overall goal, termed <GOAL>. Next we project the subgoal, in this case g, and the which tactic was used for this goal, which in this case is rule impI. We can then generate the overall skeleton:

```isar
lemma "<GOAL>"
proof -
  ... from g show ?thesis by (rule impI)
qed
```

Here, ... is the proof of g. This is generated recursively as follows. We separate between these cases:

1. the node is still open
2. the step is backwards but did not generate subgoals
3. the step is backwards and generated subgoals
4. the step is forward

Isabelle stores information relating to which type of reasoning step is used in the resolution of a goal, and we use this information to identify which one is represented by a node. We also look for any fixed variables and local assumptions at a node.

When the proof of a subgoal does not depend on any local assumptions, it is evaluated with a backward reasoning step. Cases where these steps do not generate further subgoals are usually the final steps of a proof, and are proved using the assumption tactic. If any further subgoals are generated we evaluate the nodes which represent them in the same way as the root and project them in a similar way.

If a proof is dependent on an assumption it will be discharged using a forward step. It is possible for more than one subgoal to be taken from an assumption by using different tactics, as in the script in Figure 1. Again, we reevaluate the nodes which represent these subgoals and generate the structure as before. It can be the case that a single forward step will be followed by a sequence of backward steps, or vice versa.

References

