Chapter 7

Related work

7.1 Other projects using Elpi

There have been several projects which have used Elpi to creates commands and tactics. Both Derive [Tas19] and Hierarchy Builder [CST20] center around creating definitions and do not involve creating tactics. The project Trocq [CCM24] creates commands and a tactic to facilitate proof transfer in Coq. They, do not focus on creating modular proof generators and only generate one large proof term.

7.2 Inductive predicates in program verification systems

We will discuss the different approaches to program verification and how they represent inductive predicates. There are several approaches to program verification used in the last 30 years. They can be roughly categorized into three categories when looking at inductive predicates. Program verifiers which don't use separation logic. Program verifiers which use separation logic in their own verifier. And, program verifiers which embed separation logic in an interactive proof assistant.

Program verifier without separation logic Program verifiers in this category are for example Dafny [Lei10], and Spec# [BLS04; LM10]. These program verifiers do not have to deal with representation predicates and thus do not create inductive predicates.

Separation logic program verifiers without a proof assistant These program verifiers do not have to embed the separation logic into another logic. Thus, they add inductive predicates and induction as axioms to the separation logic. Projects in this category are VeriFast [Jac+11], Viper [MSS16; SM18], and Smallfoot [BCO05].

Separation logic program verifier in a proof assistant These program verifiers embed the separation logic into the logic of the proof assistant. This can be done in several ways. Both work by Appel and Rocquencourt [AR], and Rouvoet, Krebbers, and Visser [RKV21], embed separation logic as propositions from a concrete heap to the proof assistant propositions. Thus, they can both use the inductive definition components of the respective proof assistant for defining inductive predicates in the separation logic.

The work by Chlipala [Chl11] and Bengtson, Jensen, and Birkedal [BJB12], both embed a separation logic in Coq. Both use embedding of separation logics which don't allow for using the Coq `Inductive` statement. They only support defining representation predicates using the Coq `Fixpoint`. Thus, using structural recursion.

Lastly Appel [App14] defines inductive predicates similarly to Iris. Thus, they define a monotone pre fixpoint function and take the fixpoint.

7.3 Other implementations of the IPM

In this thesis we reimplemented several tactics of the IPM. This replication of [KTB17] has been done several times before in the meta programming languages LTac2 and MTac2. And in the proof assistant Lean. The implementation in LTac2 was done in the master thesis of Liesnikov [Lie20]. They keep the same structure in their tactics as the IPM while also adding some tactics of their own.

The MTac2 meta programming language creates fully typed tactics. In the paper introducing MTac2 by Kaiser et al. [Kai+18] several tactics of the IPM where reimplemented in MTac2. This implementation focussed on showing the capabilities of MTac2 by solving several known faults in the original IPM.

Lastly, the IPM was also implemented in Lean by König [Kön22]. Unlike the previous two reimplementations of the IPM, this instance also had to replicate all definitions and lemmas since it uses a different proof assistant as its base logic.

All three reimplementations of the IPM did not consider inductive predicates. The first two reimplementations can make use of the same strategy of defining inductive separation logic predicates as used in Iris. The last reimplementations of the IPM can make use of the Lean structural recursion or a similar fixpoint construction as in Iris, to define inductive predicates.

7.4 Algorithms based on proper elements and signatures

The concept of proper elements and signatures was taken from the work by Sozeau [Soz09]. They use proper elements and signatures (called Propers in their work), to create a tactic for generalized rewriting in Coq. This tactic extends the existing `rewrite` tactic from Coq by allowing one to rewrite lemmas under terms for which an appropriate `Proper` instance is given.

This is a fairly different use of the same base definitions of signatures, and respectful, and pointwise relations. But, it informed our approach to automatically proving monotonicity pre fixpoint functions.