Chapter 7

Related work

This thesis is related to other works in several aspects. There have been multiple program verification systems that support inductive predicates, which are discussed in section 7.1. In section 7.2, we discuss other works using Elpi in Coq to develop commands and tactics. In section 7.3, we discuss reimplementations of the IPM using other meta-programming languages. Lastly, in section 7.4, we relate our algorithm, which proves the monotonicity of pre fixpoint functions, to other algorithms using signatures and proper elements.

7.1 Inductive predicates in program verification systems

We will discuss the different approaches to program verification and how they represent inductive predicates. There have been various approaches to program verification used in the past 30 years. They can be roughly categorized into three categories when looking at inductive predicates. Program verifiers that do not use separation logic. Program verifiers that use separation logic, but in their own verifier. And program verifiers that embed separation logic in an interactive proof assistant. Program verifiers which do not use separation logic are not relevant for this thesis, and thus we will start at the second category.

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 works by Appel [App06], 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. We give an example of this process below.

Example 7.1

They choose a concrete type as the propositions for the embedding of the separation logic in Coq. Given a type of heaps `heap` , the separation logic proposition is defined as

```Coq
Definition sProp := heap -> Prop.
```

Next, they can define an inductive predicate using `sProp`

```Coq
Inductive is_MLL : val -> list val -> sProp := ⋯
```

This can be unfolded to the following defintion

```Coq
Inductive is_MLL : val -> list val -> heap -> Prop := ⋯
```

This is of course definable in Coq.

Thus, the inductive predicates in the Coq logic can directly be used to define inductive predicates in the separation logic.

The work by Chlipala [Chl11] and Bengtson, Jensen, and Birkedal [BJB12], both embed a separation logic in Coq. They use embeddings of separation logics, but only make use of the Coq `Fixpoint` when defining representation predicates. Thus, they only use structural recursion.

7.2 Other projects using Elpi

There have been several projects that have used Elpi to create 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. However, all these projects fully create a proof term without elaborating in between. While we employ backwards reasoning in our proof generators and built up the proof by elaborating the proof term in between steps.

7.3 Other implementations of the IPM

In this thesis, we reimplemented several tactics of the IPM. This replication of [KTB17] has been done various times before in the meta programming languages Ltac2 and Mtac2, and in the proof assistant Lean [Mou+15].

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] some tactics of the IPM were reimplemented in Mtac2. This implementation focused on showing the capabilities of Mtac2 by making the tactic implementation more robust.

Lastly, the IPM was also reimplemented in Lean by König [Kön22]. Unlike the previous two reimplementations of the IPM, this instance had to replicate all definitions and lemmas, since it uses a different proof assistant with a different 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 utilize the Lean structural recursion or a similar fixpoint construction as in Iris to define inductive predicates. To implement our system into the Lean implementation of the IPM, one would have to use the Lean metaprogramming system. This however, does not fully follow the same ideas as Elpi, and thus we expect it to present several new challenges in the implementation.

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, i.e., rewriting with arbitrary preorders, instead of just equality. 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 of pre fixpoint functions.