Chapter 1

Introduction

A significant part of computing science has the goal of minimizing bugs in software. Formal verification systems are a promising approach towards this goal. They allow one to formally prove that a program abides by a specification. Separation logic [ORY01; Rey02] has been a promising basis for formal verification systems. It represents the state of the heap using a logic with extra connectives. The formal verification system we use on top of separation logic is Iris [Jun+15; Jun+16; Kre+17; Jun+18]. Iris has recently been used to formally verify properties of Scala [Gia+20]. It is also used in to ongoing project to formally verify the programming language Rust, called RustBelt [Jun+17; Dan+19; Mat+22].

Iris is implemented in Coq in what is called the Iris Proof Mode (IPM) [KTB17; Kre+18]. It allows for interactive verification of the specifications of the programs. In this thesis we are interested in programs involving recursive data structures, such as linked lists. Reasoning about recursive data structures in a separation logic involves reasoning with inductive predicates. They allow one to represent the heap containing the recursive data structure using constructs in the separation logic.

Unfortunately, defining inductive predicates in Iris is a very manual and labor-intensive task. Several trivial proofs must be performed, and several intermediate objects must be defined. Furthermore, using the inductive predicates in a proof requires additional manual steps.

This thesis aims to solve this problem by adding several commands and tactics to Coq that simplify and streamline working with inductive predicates. We implement our commands and tactics in the λProlog language Elpi [Dun+15; GCT19]. Elpi implements λProlog [MN86; Mil+91; BBR99; MN12] with some additions. To use Elpi as a Coq meta-programming language, there exists the Elpi Coq connector, Coq-Elpi [Tas18].

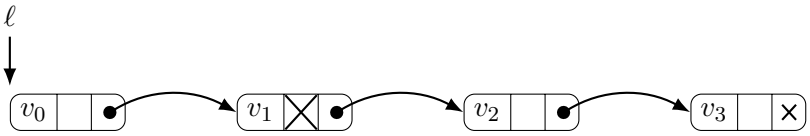

Figure 1.1: A node is shown here as three boxes next to each other, the first box contains a value. The second box is a boolean. The box is true, thus marked, if it is crossed out. The third box is a pointer, denoted by either a cross, a null pointer, or a circle with an arrow pointing to the next node.

1.1 Central example

The central example used in this thesis is *marked linked lists*, (MLLs). We will use it here to give a preview of system we developed.

MLLs, are linked lists where each node has an additional mark bit. When a node is marked, and thus the bit is set, the node is considered deleted. An example of a MLL can be found in figure 1.1. An MLL allows for deleting a node out of a list without modifying any of the other nodes, helping with concurrent usages. MLLs are an intermediary data structure in a paper by Fomitchev and Ruppert [FR04].

In order to reason about MLLs in separation logic, we relate a heap containing an MLL to a list in the separation logic in the IPM. Using our newly developed system, this can be achieved similarly to writing any other inductive predicate.

```Coq
eiInd
Inductive is_MLL : val → list val → iProp :=
    | empty_is_MLL : is_MLL NONEV []
    | mark_is_MLL v vs l tl :
      l ↦ (v, #true, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) vs
    | cons_is_MLL v vs tl l :
      l ↦ (v, #false, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) (v :: vs).
```

The command `eiInd` is prepended to the inductive statement and defines the inductive predicate `is_MLL`, together with the unfolding lemmas, constructor lemmas, and induction lemma. When we have a goal requiring induction on an `is_MLL` statement, we can simply call the `eiInduction` tactic on it. We then get goals for all the cases in the inductive predicate with the proper induction hypotheses.

1.2 Contributions

This thesis contains the following contributions.

Generation of Iris inductive predicates We develop a system written in Elpi which, given an inductive definition in Coq, defines the inductive predicate with associated unfolding, constructor and induction lemmas. In addition, tactics are created which automate unfolding the inductive predicate and applying the induction lemma. *(Ch. 5)*

Modular tactics in Elpi We present a way to define steps in a tactic, called *proof generators*, such that they can easily be composed. Allowing one to define simple proof generators which can be reused in many tactics. *(Sec. 4.7)*

Generate monotonicity proof of n-ary predicates We present an algorithm which given an n-ary predicate can find a proof of monotonicity. *(Sec. 3.3)*

Evaluation of Elpi Lastly, we evaluate Elpi with Coq-Elpi as a metalanguage for Coq. We also discuss replacing LTaC with Elpi in IPM. *(Ch. 6)*

1.3 Outline

We start by giving a background on Separation logic in chapter 2thesis.pdf. The chapter discusses the Iris separation logic while specifying and proving a program on MLLs. Next, in chapter 3thesis.pdf, we discuss defining representation predicates in a separation logic using least fixpoints. Thus, we show how to define a representation predicate as an inductive predicate, and then give a novel algorithm to prove it monotone. In chapter 4thesis.pdf, we give a tutorial on Elpi by implementing an IPM tactic, `iIntros`, in Elpi. Building on the foundations of chapter 4thesis.pdf, we create the command and tactics to define inductive predicates in chapter 5thesis.pdf. In chapter 6thesis.pdf, we evaluate what was usefull in Elpi and what could be improved. We also discuss how and if Elpi can be used in IPM. Lastly, we discuss related work in chapter 7thesis.pdf and show the capabilities and shortcomings of the created commands and tactics in chapter 8thesis.pdf, together with any future work.

During the thesis, we will be working in two different programming languages. In order to always distinguish between them, the inline displays have a different color. Any `Coq displays` have a light green line next to them. Any `Elpi displays` have a light blue line next to them. Full-width listings also differentiate using green and blue lines respectively.