Abstract

Reasoning about programs that involve shared imperative data structures has proven to be notoriously difficult over the years, mainly due to problems related to aliasing. A recent approach to these problem was presented by O'Hearn and Reynolds; they devised a program logic in the style of Hoare, called *separation logic*, and that is the main interest of this thesis. Here is a brief overview of the main contributions.

Since separation logic is a relatively new logic, we should be able to "benchmark" it on realistic examples. We show that separation logic is indeed useful by specifying and proving correctness of Cheney's copying garbage collector, a program which uses sophisticated pointer manipulations. The main contribution of the work is an illustration of how additions of sets and relations to the languages of expressions and assertions can be used in combination with ideas from separation logic to specify and prove an isomorphism between the states before and after execution of the algorithm.

The assertion language of traditional separation logic is a variant of first order propositional BI. We make a straightforward extension of this logic to *higher-order predicate BI*, and demonstrate how it can be used in separation logic. In particular, we show that the extension can be applied to model data abstraction and polymorphism for a programming language with heap manipulating constructs and calls to simple procedures. Moreover, we present a model for this new logic (called *higher order separation logic*), using a general class of models called *BI hyperdoctrines*.

The programming language of standard separation logic is a simple extension of Hoare's **while**-language with basic heap manipulating constructs. The success of separation logic relies heavily on its ability to support *local reasoning* via the so-called *frame rule*. We extend the scope of separation logic by devising a *separation-logic type system* for a version of idealized algol with heap manipulating constructs, allowing for local reasoning about higher-order programs which manipulate heaps. Moreover, we present a categorical model in which one can soundly and coherently model the programming language in a way that is adequate with respect to the standard semantics of idealized algol. We also take some initial steps towards a *parametric* model for separation-logic typing.

In Hoare's work on *refinement* for imperative programs one starts with an abstract specification of a data type and derives a concrete representation. This method assumes a static separation of variables between representations of the data type; however, the introduction of pointers into refinement breaks these assumptions. We introduce a model that brings ideas from separation logic into refinement, and prove an abstraction theorem for so-called *separation contexts* which, intuitively, are client programs that are well-behaved with respect to resources.