New Foundations for Separation Logic

Hiep, H.A.

Chapter 5

Conclusion

In this thesis, we investigated new foundations for separation logic. These new foundations are presented in two parts: in the first part, we presented a model theoretic investigation of separation logic with the aim of finding an adequate semantics, and a novel finitary proof system for separation logic for which we showed soundness and completeness. In the second part, we proposed a new interpretation of Reynolds' logic in such a way that it is compatible with the semantics of the first part: it turned out that all axioms are still sound and relatively complete with respect to the new interpretation. We also introduced dynamic separation logic, and showed that it yields an alternative axiomatization of Reynolds' logic that avoids introducing separating connectives when generating weakest preconditions and strongest postconditions.

Summary

To see more clearly the bigger picture, we summarize this thesis in Table 5.1. The table displays the different subjects studied and references various sections. As a starting point of this thesis we considered background knowledge, shown in the left column of Table 5.1: first-order logic, Hoare's logic, and dynamic logic.

- First-order logic is a well-known assertion language with a rich model theory and proof theory, and Gödel's completeness theorem connects these two views on first-order logic. Chapter A of the appendix revisits the necessary background knowledge on first-order and higher-order logic.

- Hoare's logic is a program logic useful for reasoning about program correctness. A program S is provably correct with respect to a specification that specifies the precondition ϕ and postcondition ψ, denoted by the Hoare triple $\{\phi\}\ S\ \{\psi\}$, if it is the case that such Hoare triple can be deduced from Hoare's axioms. The proof system of Hoare's logic can be given a rich program semantics, for which it can be shown that there is a soundness and relative completeness result. Chapter B of the appendix revisits the necessary background knowledge on Hoare's logic, but presents it in a general way that is more suitable for the rest of this thesis.

First-order logic (FOL)

- well-known assertion language (see Section A.1)
- rich model theory (see Section A.2)
- rich proof theory (see Section A.3)
- Gödel's completeness theorem (see Section A.4)

Separation logic (SL)

- extends FOL with connectives $*$, $-\!\!*$ (see Section 2.1)
- *model theoretic investigation* (see Section 2.2, 2.3)
- *general semantics and proof theory* (see Section 3.1, 3.3)
- *soundness and completeness* (see Section 3.2, 3.4)

Hoare's logic (HL)

- Hoare triple $\{\phi\}\ S\ \{\psi\}$ (see Section B.1)
- rich program semantics (see Section B.2, B.3)
- Cook's relative completeness (see Section B.4)

Reynolds' logic (RL)

- *semantics relative to memory models* (see Section 4.1)
- *richer* program semantics (see Section 4.2)
- *soundness and relative completeness* (see Section 4.3)

Dynamic logic (DL)

- extends FOL with modality $[S]\phi$
- HL is embedded in DL
- DL is more expressive than HL

Dynamic separation logic (DSL)

- *extends DL with connectives* $*$, $-\!\!*$ (see Section 4.4)
- *RL is embedded in DSL* (see Section 4.5)

Table 5.1: One-page summary of this thesis. Novel contributions are *emphasized*.

- Dynamic logic is an extension of first-order logic, by adding a so-called program modality to the assertion language. It is the case that Hoare's logic can be embedded in dynamic logic: every program that can be proven to be correct in Hoare's logic also can be proven to be correct in dynamic logic. However, dynamic logic is more expressive than Hoare's logic. The reader can consult [107] for more background knowledge on dynamic logic.

This thesis continued study of the subjects shown in the right column of Table 5.1: separation logic, Reynolds' logic, and dynamic separation logic. The first contribution is nominal, by disambiguating *separation logic* from *Reynolds' logic* (where the former strictly speaks of the logic of the assertion language, and the latter speaks of the program logic).

- Separation logic is an extension of first-order logic that introduces two new connectives. We recalled the syntax of separation logic, and have shown that its standard semantics is inadequate. Searching for an adequate semantics has given several interesting results: full separation logic is inadequate, and several fragments of the syntax of separation logic are also inadequate. We then have introduced a more general semantics, akin to Henkin's general semantics, for separation logic and defined two proof systems for deducing consequences: one sequent calculus, and one natural deduction system. We have shown that for both we have soundness and completeness, and this settles the open problem of adequacy for separation logic (the separation logic counterpart to Gödel's completeness theorem connecting model theory and proof theory).

- Reynolds' logic is an extension of Hoare's logic in two ways: Hoare triples $\{\phi\}\ S\ \{\psi\}$ now speak about pointer programs S too, and the precondition ϕ and postcondition ψ are now assertions of separation logic. All Hoare triples provable from Hoare's axioms are also provable in Reynolds' logic, but Reynolds' logic proves more triples than in Hoare's logic (e.g. those involving separating connectives). We have given a general semantics to Reynolds' logic relative to memory models, and we have based the program semantics on top of the semantics of the primitive operations of pointer programs: lookup, mutation, allocation, deallocation. Reynolds' axioms, and the frame rule in particular, are still sound and relatively complete in the more general setting, even when we allow for infinite heaps.

- Dynamic separation logic is an extension of first-order dynamic logic by adding the separating connectives, similar to how separation logic adds the separating connectives to first-order logic. First-order dynamic separation logic is a novel subject, not studied in the literature before. We have seen how to give semantics to dynamic separation logic, and we have investigated a calculus for reducing program modalities involving the primitive pointer-manipulating operations. Since it is also the case that Reynolds' logic can be embedded in dynamic separation logic, this resulted in an alternative axiomatization of Reynolds' logic that analyzes the logical structure of pre- or postconditions.

Future work

In this section, we discuss suggestions of directions for further study. One direction is to systematically compare the different semantics for separation logic that were introduced in this thesis. The next direction is to fully formalize the results presented in this thesis in a proof assistant, and the soundness and completeness of the proof systems for separation logic in particular, to increase our confidence in the results. The third direction is to enhance tool support for reasoning about separation logic. Another direction is to investigate relative completeness in the case of recursive procedures in such a setting that the frame rule is necessary. Further directions are investigating other separation logic variants, such as intuitionistic separation logic and permission-based separation logic, and other program logics, such as concurrent separation logic. A last direction is to investigate the Curry-Howard correspondence for our new natural deduction proof system for separation logic, and whether our new semantics also works with proof systems such as the logic of bunched implications.

Different semantics

In the chapters introducing new semantics for separation logic and Reynolds' logic, we have seen the following interpretations of separation logic:

- **SSL** for Standard Separation Logic which interprets separation logic formulas with respect to all finite heaps (see Definition 2.2.1),
- **FSL** for Full Separation Logic which interprets separation logic formulas with respect to all (finite or infinite) heaps (see Definition 2.3.1),
- **GSL** for General Separation Logic which interprets separation logic formulas with respect to a fixed set of (finite or infinite) heaps (see Definition 4.1.2),
- **MSL** for Memory Separation Logic which interprets separation logic formulas with respect to memory models, a set of heaps that satisfies a number of closure conditions (see Definition 4.1.6) necessary for showing the soundness and relative completeness of Reynolds' logic.

The valid formulas of separation logic are included in the following ways:

- the set of valid formulas according to **GSL** are included in the set of valid formulas according to **MSL**,
- the set of valid formulas according to **MSL** are included in the set of valid formulas according to **SSL** and also in the set of valid formulas according to **FSL**.

So, it is possible to see **MSL** as a generalized interpretation that does not depend on the finiteness condition of the heap.

To obtain a sound and complete proof system, we relaxed the condition that the heap is functional (in the sense of a functional relation) to obtain the following

relational separation logic interpretations (not to be confused with relational separation logic as investigated by Yang [231]):

- **WRSL** for Weak Relational Separation Logic which interprets separation logic formulas with respect to relational structures with respect to all finite relations (see Definition 2.5.1),
- **FRSL** for Full Relational Separation Logic which interprets separation logic formulas with respect to relational structures with respect to all relations (see Definition 2.5.2),
- **GRSL** for General Relational Separation Logic which interprets separation logic formulas with respect to relational structures and a fixed set of relations (see Definition 3.4.2),
- **RSL** for Relational Separation Logic which interprets separation logic formulas with respect to comprehensive relational structures (see Definition 3.4.4) necessary for obtaining an adequate and finitary proof system.

The interpretations of **SSL**, **FSL** and **GSL** can be embedded in a natural way in the interpretations of **WRSL**, **FRSL** and **GRSL**, respectively.

The interpretations **RSL** and **MSL** are important, since the finitary proof system for separation logic we present in this thesis is sound and complete with respect to **RSL**, whereas Reynolds' logic is sound and relatively complete with respect to **MSL**. Hence, by taking the intersection of these two interpretations, we obtain our final desired interpretation of separation logic and Reynolds' logic.

Since it is possible to instantiate Reynolds' logic with different interpretations of separation logic (such as the standard interpretation, full interpretation, or general interpretations such as those restricted to first-order definable heaps and memory models), each interpretation of separation logic induces a different set of valid formulas which can be used in the consequence rule. This situation is similar to that of the different interpretations of higher-order logic (weak interpretation, full interpretation, Henkin interpretation), where each interpretation induces a different set of valid formulas. The same formula can thus be interpreted differently, and it remains future work to study how these interpretations are related. It also remains future work to study the semantic relation between the different instances of Reynolds' program logic with different interpretations of separation logic.

Formalization

From the position of the formalist, who rejects meaning and truth in mathematics that transcends literal symbols and their formation rules, much of mathematics can be only understood by being based on some axiomatic treatment of set theory (or any other foundational theory). As such, a piece of mathematics is only valid to the formalist in so far that it can be formalized and shown to follow from axioms. The ideal formalist never skips any detail and never needs an appeal to intuition.

Although much formal detail is provided, also in this thesis some details are only sketched out and there were some appeals to intuition. To further increase

confidence in results, one could formalize mathematical statements using a proof assistant and meticulously check the argument by also formalizing its proof and filling in all the details. As such, we have indeed formalized important definitions and results of Chapter 3 and Chapter 4 in the Coq proof assistant, see Chapter D in the appendix. However, not all results presented in this thesis have been formalized and checked by a proof assistant, and as such there remains a threat to validity.

Another direction of future work is to formalize the semantics of separation logic and our suggested proof systems, and verify the soundness and completeness result by means of a proof assistant. Such future work could increase confidence in the claimed results of this thesis.

Tool support

Since the alternative axiomatization of Reynolds' logic we arrived at by introducing dynamic separation logic and the original axiomatization given by Reynolds himself are both weakest precondition axiomatizations, we have that they must be equivalent. However, evaluating existing tools for reasoning about separation logic on simple programs and postconditions, on formulas that express the equivalence between Reynolds' weakest precondition and our alternative weakest precondition, results in bugs or incompleteness.

Recall from the introduction that we have seen the generation of two weakest preconditions for the program $[x] := 0$ and the postcondition $(y \hookrightarrow z)$:

$$(x \mapsto -) * ((x \mapsto 0) \mathrel{-\!\!*} (y \hookrightarrow z)) \tag{1.1}$$

$$\equiv$$

$$[[x] := 0](y \hookrightarrow z) \tag{1.2}$$

$$\equiv$$

$$(x \hookrightarrow -) \wedge ((y = x \wedge z = 0) \vee (y \neq x \wedge y \hookrightarrow z)) \tag{1.3}$$

We have that (1.1) and (1.3) are equivalent. Surprisingly, a proof of the equivalence exceeds the capability of all the automatic separation logic provers in the benchmark competition for separation logic [201]. In particular, of the automatic provers, only the CVC4-SL tool [186] supports the fragment of separation logic that includes the separating implication connective. However, from our own experiments with that tool, we found that it produces an incorrect counter-example and reported this as a bug to one of the maintainers of the project (Andrew Reynolds). In fact, the latest version, CVC5-SL, reports the same input as 'unknown', indicating that the tool is incomplete. In the case of (semi-)interactive separation logic provers or proof assistants (such as Iris [132], and VerCors [12, 156] that uses Viper [159] as a back-end) we sought out expertise and collaborated in our search for a tool-supported proof of the above equivalence. Even after personally visiting the Iris team in Nijmegen (lead by Robbert Krebbers) and the VerCors team in Twente (lead by Marieke Huisman), we were unable to guide the tools to produce a proof of equivalence. The problem here seems similar to that of [122], in that their semantics of the separating connectives, which are formalized in terms of

abstract monoids, are not compatible with the set-theoretic interpretation of the points-to relation. However, only further investigation can explain what is the reason existing tools contain bugs or are incomplete.

Another direction of future work is to develop proof assistants and automatic provers based on the novel semantics and proof theory introduced in this thesis. A concrete direction of future study is to improve the KeY system, which is based on a variant of dynamic logic supporting Java programs called JavaDL, to extend its implementation of dynamic logic to also work with separation logic formulas, which could improve the efficiency for reasoning about aliasing in Java programs.

Recursive procedures

In this thesis we have only shown the relative completeness result of Reynolds' logic for the primitive operations of pointer programs. Our result can be readily extended to also cover relative completeness for the entire programming language, including the complex programs, as in [81, 209]. In fact, in the relative completeness result of [81], there is no need for the frame rule by arithmetically encoding finite heaps. It remains future work to show how to extend Reynolds' proof system to recursive procedures, in which it is possible to give a relative completeness result that is based on the frame rule and does not depend on an encoding of the heap (preliminary results are submitted to a conference for peer review).

Other separation logics

The results presented in this thesis are based on the classical interpretation of separation logic. There is also an intuitionistic interpretation of separation logic [187], based on which one could conceive a corresponding and novel intuitionistic dynamic separation logic. In fact, preliminary results have already indicated that much of the work presented in this thesis can be repeated: a novel alternative weakest precondition for intuitionistic separation logic can be defined. In fact, by following the approach of this thesis, also a novel strongest postcondition can be given for intuitionistic separation logic—which, as far as the author knows, has not yet been given in the literature before. These preliminary results are presented in Appendix C.

Also there are many variants of separation logic, such as probabilistic separation logic [16], permission-based separation logic [33], set-based separation logic [110], strong-separation logic [172]. It remains future work to investigate how to adapt the semantics introduced in this thesis to those different settings.

Other program logics

O'Hearn and Brookes have introduced concurrent separation logic, which is an extension of Reynolds' logic for reasoning about concurrent shared-memory pointer programs [163, 36, 37, 38]. The logic introduced by O'Hearn and Brookes is a program logic: they use the same assertion language, separation logic, as is investigated in this thesis. Their approach is based on the Owicki-Gries proof

method [171] published in 1976. There were two papers published by S. Owicki and D. Gries in 1976, one published by the ACM [171] and the other published by Springer [170]. Both papers were based on the work presented in Owicki's Ph.D. thesis. The ACM paper explains her work in a more simplified setting, and makes use of resources which protect program variables and ensures that the execution of critical sections are mutually exclusive, i.e. never two critical sections are executed at the same time. This is also the starting point for O'Hearn and Brookes when introducing concurrent separation logic.

The Springer paper, however, offers a more 'primitive tool, so primitive that other methods for synchronization such as semaphores and events can be easily described using it.' The Springer paper is more general and allows 'to prove correctness for programs of such a fine degree of interleaving that the only mutual exclusion need be the memory reference.' [170] In this general setting, proving correctness involves showing that parallel programs are interference free, by using an interference freedom test on the proof outlines of the parallel components. In contrast, interference freedom is a property that comes for free by using resources and certain syntactic restrictions, as presented in the ACM paper: 'complexity of parallel programs stems from the way processes can interfere with each other as they use shared variables. The critical section statement reduces these problems by guaranteeing that only one process at a time has access to the variables in a resource. The following syntax restrictions ensure that...' [171]

It remains future work to see whether Reynolds' logic as presented in this thesis can also be extended to concurrent pointer programs, where the only mutual exclusion that is provided by the programming language is at the level of accessing or mutating a single memory reference and interference freedom tests now involves checking proof outlines where assertions are formulated in the language of separation logic (preliminary results are submitted to a conference for peer review). Also other program logics, such as separation logic with higher-order store [28], can be investigated.

Curry-Howard correspondence

A well-known connection between logic and computation, the Curry-Howard correspondence, is often summarized by the slogans 'propositions as types' and 'programs as proofs' [220]. In general, one speaks of a Curry-Howard correspondence whenever it is possible to relate proofs in a proof theory on the one hand with terms in a calculus on the other hand, in such a way that proof normalization of the proof theory corresponds with term reduction in the calculus.

In this thesis we have introduced a natural deduction proof system for reasoning about separation logic. Further work can be done in investigating meta-properties of this proof system, such as proof normalization and normalization strategies. One could investigate this by asking the question: what are the terms and rewrite rules of the lambda-like calculus such that the Curry-Howard correspondence holds with respect to the natural deduction proof system we introduced in this thesis?

Furthermore, one could investigate existing proof systems for separation logic,

such as the proof system of bunched implications by O'Hearn and Pym [167, 181]. Is it possible to adapt existing proof systems to work with the Henkin-like semantics introduced in this thesis, and is it also there possible to obtain soundness and completeness results? What are the ramifications of our novel semantics to existing work [88] that investigates the Curry-Howard correspondence for the proof system of bunched implications?