A Separation Logic Framework in HOL

Thomas Tuerk
University of Cambridge, Computer Laboratory

UNIVERSITY OF CAMBRIDGE

Separation Logic

Separation logic is an extension of Hoare logic that allows local reasoning about mutable data structures. It's been introduced by O'Hearn, Reynolds and Yang in 2001.

$$\frac{\{P\}\ \mathit{prog}\ \{Q\}}{\{P * R\}\ \mathit{prog}\ \{Q * R\}} \qquad \frac{\{P_1\}\ \mathit{prog}_1\{Q_1\} \qquad \{P_2\}\ \mathit{prog}_2\{Q_2\}}{\{P_1 * P_2\}\ \mathit{prog}_1 || \mathit{prog}_2\ \{Q_1 * Q_2\}}$$

There are several implementations: Smallfoot [1], Slayer and Spaceinvader are probably some of the best know examples. There are also formalisations inside theorem provers.

Proposed Framework

The implementations mentioned above all focus on one specific programming language (mostly some C-like imperative language). There are good reasons for this, however it makes reusing these formalisations very difficult and distracts from the core features of separation logic.

Therefore, I suggest building a framework for separation logic inside the HOL theorem prover that concentrates on separation logic itself instead of a concrete programming language. Concrete programming languages can be handled using instantiations of the framework.

Following this approach, I hope to gain twice: the general part can ignore low-level details and thus be cleaner and easier to understand. Moreover, this additional layer of abstraction allows reuse in a natural way.

Initial steps

I believe, that *Abstract Separation Logic* [2] is a good starting point to build such a framework. I formalised it in HOL. Then, I started building a tool similar to Smallfoot based on it as a first case study (see `http://hol.sourceforge.net`, subdirectory `examples/separationLogic`). This tool uses a programming language very similar to the one used by Smallfoot. However, it incorporates some ideas from *Variables as Resource in Hoare Logic* [3] as well. The tool is able to parse Smallfoot specifications and verify their correctness completely automatically. However, not all features of Smallfoot are supported yet.

Abstract Separation Logic

Most separation logics operate on states consisting of a stack and a heap. In contrast, abstract separation logic can use arbitrary states. A partial function $\bullet$ is used to combine these states. Two states s_1 and s_2 are *separate*, iff $s_1 \bullet s_2$ is defined. This combination function $\bullet$ has to satisfy some properties: a neutral element u has to exist, such that $(\bullet, u)$ is a cancellative, partial commutative monoid. Using this notion, one can easily define the spatial conjunction operator $*$ as follows:

$$P * Q := \{s \mid \exists p, q.\ (p \bullet q = s)\ \wedge\ p \in P\ \wedge\ q \in Q\}$$

Other standard separation logic constructs can be defined in a natural way as well. By instantiating $\bullet$ one can instantiate the framework to the logic used for a specific programming language. Abstract separation logic contains also the definition of an abstract, imperative programming language.

Smallfoot

Smallfoot [1] is a tool that can automatically verify specifications of programs written in a simple, imperative language. It uses light-weight specifications about dynamically allocated pointer structures on a heap that are expressed in a simple fragment of separation logic:

$$\begin{array}{ccc} e_1 \doteq e_2 & e_1 \neq e_2 & \mathit{emp} \\ e \mapsto [t_1 : e_1, \ldots, t_k : e_k] & \mathit{list\_seg}(tl, e_1, e_2) & \mathit{sf}_1 * \mathit{sf}_2 \\ \ldots & & \end{array}$$

Example

```
list_remove(l;x) [list(l)] {
  local t;
  if(l!=NULL) {
    if(l==x) {
      l = l->tl;
      dispose(x);
    } else {
      t = l->tl;
      list_remove(t;x);
      l->tl = t;
    }
  }
} [list(l)]
```

Smallfoot is able to verify this annotated function definition completely automatically. So is my HOL implementation of Smallfoot.

Conclusions and Future Work

- I propose to build a framework for separation logic based on abstract separation logic [2] that can be instantiated to different programming languages / flavours of separation logic
- I have formalised abstract separation logic.
- As a first case study, a tool similar to Smallfoot was build.
- I plan to extend this tool to allow the interactive verification of of more complicated properties.
- The framework should be extended to allow more intuitive definitions of some programming language features.
- Other case studies are planned.

Acknowledgements

I thank Matthew Parkinson, Mike Gordon, Alexey Gotsman, Magnus Myreen and Viktor Vafeiadis for a lot of discussions, comments and criticism.

References

[1] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO*, pages 115–137, 2005.

[2] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 366–378, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Matthew Parkinson, Richard Bornat, and Cristiano Calcagno. Variables as resource in hoare logics. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 137–146, Washington, DC, USA, 2006. IEEE Computer Society.