Holfoot - A Formalisation of Smallfoot in HOL

Thomas Tuerk
University of Cambridge, Computer Laboratory

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.

Separation Logic 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 have build a framework for separation logic inside the HOL theorem prover that is based on *Abstract Separation Logic* [2] (sources at `http://hol.sourceforge.net`, subdirectory `examples/separationLogic`).

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.

Holfoot

As a first case study, I instantiated the framework to build a tool called *Holfoot*. It's used programming language and specifications are similar to Smallfoot [1]. However, it incorporates some ideas from *Variables as Resource in Hoare Logic* [3] as well. Holfoot is able to parse Smallfoot specifications and verify their correctness fully automatically.

In addition to Smallfoot, Holfoot can reason about the content of datastructures as well as their shape. This allows the verification of fully functional specifications. Simple algorithms like list-copy, list-reverse or list-length can still be handled automatically, however the verification of algorithms like mergesort needs interaction.

Examples

```
list_copy(z;c) [data_list(c,data)] {
  local x,y,w,d;
  if (c == NULL) { z=NULL; }
  else {
    z=new(); z->tl=NULL; x = c->dta; z->dta = x; w=z; y=c->tl;
    while (y != NULL) [data_lseg(c,``_data1++[_cdate]``,y) * data_list(y,_data2) *
                       data_lseg(z,_data1,w) * w |-> tl:0,dta:_cdate *
                       ``data:num list = _data1 ++ _cdate::_data2``] {
      d=new(); d->tl=NULL; x=y->dta; d->dta=x; w->tl=d; w=d; y=y->tl;
    }
  }
} [data_list(c,data) * data_list(z,data)]
```

```
list_reverse(i;) [data_list(i,data)] {
  local p, x;
  p = NULL;
  while (i != NULL) [data_list(i,_idata) * data_list(p,_pdata) *
                     ``(data:num list) = (REVERSE _pdata) ++ _idata``] {
    x = i->tl; i->tl = p; p = i; i = x;
  }
  i = p;
} [data_list(i,``REVERSE data``)]
```

```
list_length(r;c) [data_list(c,cdata)] {
  local t;
  if (c == NULL) {r = 0;} else {
    t = c->tl; list_length(r;t); r = r + 1;
  }
} [data_list(c,cdata) * r == ``LENGTH (cdata:num list)``]
```

These examples can be verified completely automatically. More examples can be found at `http://hol.sourceforge.net` and at `http://wiki.heap-of-problems.org`.

```
merge(r;p,q) [data_list(p,pdata) * data_list(q,qdata) *
              ``(SORTED $<= pdata) /\ (SORTED $<= qdata)``] { ...
} [data_list(r,_rdata) * ``(SORTED $<= _rdata) /\ (PERM (pdata ++ qdata) _rdata)``]

split(r;p) [data_list(p,data)] {  ...
} [data_list(p,_pdata) * data_list(r,_rdata) * ``PERM (_pdata ++ _rdata) data``]

mergesort(r;p) [data_list(p,data)] {
  local q,q1,p1;
  if (p == NULL) r = p;
  else {
    split(q;p);
    mergesort(q1;q) || mergesort(p1;p);
    merge(r;p1,q1);
  }
} [data_list(r,_rdata) * ``(SORTED $<= _rdata) /\ (PERM data _rdata)``]
```

While the program structure and the shape of data structures can be handled automatically, the user is left to reason about properties of sorted lists and permutations. However, the following short proof script is sufficient to verify mergesort:

```
val thm = smallfoot_verbose_prove(mergesort-specification-filename,
   SMALLFOOT_VC_TAC THEN
   ASM_SIMP_TAC (arith_ss++PERM_ss) [SORTED_EQ, SORTED_DEF, transitive_def] THEN
   REPEAT STRIP_TAC THEN (
      IMP_RES_TAC PERM_MEM_EQ THEN
      FULL_SIMP_TAC list_ss [] THEN
      RES_TAC THEN ASM_SIMP_TAC arith_ss []
   ));
```

Conclusions

- I have build a framework for separation logic based on abstract separation logic [2] that can be instantiated to different programming languages / flavours of separation logic
- As a first case study, Holfoot was build.
- Holfoot can be used to verify fully functional specifications.

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.