A formalisation of Smallfoot in HOL

Thomas Tuerk

19th August 2009

Separation Logic

- Separation logic is an extension of Hoare Logic
- successfully used to reason about programs using pointers
- allows local reasoning, scales nicely
- there are some implementations
  - Smallfoot (Calcagno, Berdine, O'Hearn)
  - Slayer (MSR, B. Cook, J. Berdine et al.)
  - Space Invader
  - ...
- there are formalisation in theorem provers
  - *Concurrent C-Minor Project*, Coq (Appel et al.)
  - *Practical Tactics for Separation Logic* (McCreight)
  - *Types, Bytes, and Separation Logic*, Isabelle/HOL (Tuch, Klein, Norrish)

Motivation

- there are a lot of slightly different separation logics
- all tools / formalisations I know of are designed for one specific programming language
- in contrast, I developed a **general separation logic framework**
  - concentrate on the essence of separation logic
  - high level of abstraction
  - this leads to simple definitions and proofs
  - high level of reuse by instantiation to different settings
- this framework is used to build a tool similar to Smallfoot

Outline

- **Abstract Separation Logic**
  - the core of the framework
  - contains an abstract programming language and an abstract specification language
- **Holfoot**, a formalisation of Smallfoot
  - instantiates the framework
  - parser for Smallfoot example files
  - completely automatic verification of Smallfoot examples
  - interactive proofs are possible as well
  - most features of Smallfoot are supported
  - additionally: reasoning about data content
  - thus: reasoning about fully-functional specifications

Abstract Separation Logic

- abstract separation logic is an abstract version
- introduced by Calcagno, O'Hearn and Yang in *Local Action and Abstract Separation Logic*
- abstraction helps to concentrate on the essential part
- embedding in a theorem prover becomes easier
- can be instantiated to different variants of separation logic
- therefore, it is a good basis for a separation logic framework

Introduction to Abstract Separation Logic

Separation Logic on Heaps

- heaps
- disjoint union of heaps $\uplus$
- h_1, h_2 have disjoint domains
- $h \models P_1 * P_2$ iff $\exists h_1, h_2.\ (h = h_1 \uplus h_2)\ \wedge$ $h_1 \models P_1\ \wedge\ h_2 \models P_2$

Abstract Separation Logic

- abstract states
- abstract separation combinator $\circ$
- $s_1 \circ s_2$ is defined
- $s \models P_1 * P_2$ iff $\exists s_1, s_2.\ (s = s_1 \circ s_2)\ \wedge$ $s_1 \models P_1\ \wedge\ s_2 \models P_2$

Abstract Specification Logic

- a **separation combinator** $\circ$ is a partially defined function such that:
  - $\circ$ is **associative**
    $\forall x\, y\, z.\ (x \circ y) \circ z = x \circ (y \circ z)$
  - $\circ$ is **commutative**
    $\forall x\, y.\ x \circ y = y \circ x$
  - $\circ$ is **cancellative**
    $\forall x\, y\, z.\ (x \circ y = x \circ z) \Rightarrow y = z$
  - forall elements there is a **neutral element**
    $\forall x. \exists u_x.\ u_x \circ x = x$
- the usual separation logic operators are defined using $\circ$
- predicates are sets of states

Hoare Triples and Actions

- consider **partial correctness** of **nondeterministic programs**
- elementary construct of programs are *local actions*
- given a state s an **action** can
  - fail, i. e. return a special state $\top$,
  - succeed, i. e. return a non-empty set of successor states,
  - diverge, i. e. return $\emptyset$.
- thus, actions are functions from states to $\top$ or a set of states
- **Hoare Triples** are defined as usual:

$$\{P\}\ \text{action}\ \{Q\} \iff \begin{array}{l}\forall s.\ s \models P\ \Rightarrow \text{action}(s) \neq \top \wedge \\ \qquad\qquad\quad \forall t \in \text{action}(s).\ t \models Q\end{array}$$

Local Actions / Frame Rule

Frame Rule

$$\frac{\{P\}\ \text{action}\ \{Q\}}{\{P * R\}\ \text{action}\ \{Q * R\}}$$

- frame rule is essential for separation logic
- it's important for local reasoning
- it does not hold for arbitrary actions
- actions that respect the frame rule are called **local**
- just local actions will be considered in the following

Programs

- **programs** consist of local actions
- there are
  - consecutive execution
  - conditional execution
  - loops
  - mutual recursive procedures
  - parallelism
  - semaphores
  - nondeterministic choice
- inference rules are proved for these programs
- tool support exists

Smallfoot

- "Smallfoot is an automatic verification tool which checks separation logic specifications of concurrent programs which manipulate dynamically-allocated recursive data structures."(Smallfoot documentation)
- developed by
  - Cristiano Calcagno
  - Josh Berdine
  - Peter O'Hearn
- the framework has been instantiated to build a HOL-version of Smallfoot, called **Holfoot**

Instantiation

- first step
  - add a stack with explicit read/write permissions
  - this uses ideas from *Variables as Resource in Hoare Logics* by Parkinson, Bornat and Calcagno
  - this abstract stack is sufficient for
    - pure expressions
    - assignments
    - local variables
    - call-by-value and call-by-reference arguments
    - ...
- second step
  - add a heap similar to the one used by Smallfoot
  - this allows
    - allocation / deallocation
    - heap lookups and assignments
    - predicates for lists, trees
    - ...

Examples I

mergesort.sf

```
split(r;p) [list(p)] {
  local t1,t2;
  if(p == NULL) r = NULL;
  else {
    t1 = p->tl;
    if (t1 == NULL) {
      r = NULL;
    } else {
      t2 = t1->tl;
      split(r;t2);
      p->tl = t2;
      t1->tl = r;
      r = t1;
    }
  }
} [list(p) * list(r)]

merge(r;p,q)
   [list(p) * list(q)] {
   ...
} [list(r)]

mergesort(r;p) [list(p)] {
  local q,q1,p1;
  if(p == NULL) r = p;
  else {
    split(q;p);
    mergesort(q1;q);
    mergesort(p1;p);
    merge(r;p1,q1);
  }
} [list(r)]
```

Holfoot can verify such specifications completely automatically!

Examples II

- there is a parser for such Smallfoot specification
- Smallfoot comes with a collection of examples, most of which can be verified completely automatically
  - list reversal
  - list filtering
  - list appending
  - parallel mergesort
  - ...
- more examples can be found at `http://heap-of-problems.org`
- in addition to Smallfoot, there is support for data content

Examples III

mergesort.dsf

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

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)``]

mergesort(r;p) [data_list(p,data)] { ...
} [data_list(r,_rdata) * ``(SORTED $<= _rdata) /\ (PERM data _rdata)``]
```

Examples III

mergesort.dsf

```
split(r;p) [data_list(p,data)] { ...
} [exists pdata, rdata. data_list(p,pdata) * data_list(r,rdata) *
   ``PERM (pdata ++ rdata) data``]

merge(r;p,q) [data_list(p,pdata) *  data_list(q,qdata) *
               ``(NUM_SORTED pdata) /\ (NUM_SORTED qdata)``] { ...
} [exists rdata. data_list(r,rdata) * ``(NUM_SORTED rdata) /\
   (PERM (pdata ++ qdata) rdata)``]

mergesort(r;p) [data_list(p,data)] { ...
} [exists rdata. data_list(r,rdata) *
   ``(NUM_SORTED rdata) /\ (PERM data rdata)``]
```

Examples IIII

```
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 []
   ));
```

Conclusion and Future Work

Conclusion

- I have built a general separation logic framework
- the power and flexibility of the framework is demonstrated by implementing a HOL version of Smallfoot
- Holfoot combines the power of HOL with the automation of Smallfoot
- this combination allows reasoning about data
- Holfoot can verify fully-functional specifications

Future Work

- finish a general clean-up
- extend Holfoot to reason about pointer arithmetic
- verify programs using more complicated data structures