A fully-expansive HOL implementation of Smallfoot

Thomas Tuerk

2nd September 2008

Overview

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.)
  - ...
- there as formalisation in theorem provers
  - *Concurrent C-Minor Project*, Coq (Appel et al.)
  - *Types, Bytes, and Separation Logic*, Isabelle/HOL (Tuch, Klein, Norrish)

Motivation

- there are a lot of slightly different separation logics
  - classically a state constists of stack + heap
  - but: how does the heap look like
  - read- / write-permissions for stack-variables ?
  - which predicates are supported?
- all tools / formalisations I know of are designed for one specific programming language
- in contrast, I would like to design a general framework
  - keep the core as abstract as possible
  - this should lead to simplicity
  - instantiate this core to different specific programming languages
- Main questions: what's the essence of separation logic? How to formalise it into a theorem prover?

Work done up to this point

- formalisation of *Abstract Separation Logic*
- first case study: a tool similar to Smallfoot
  - combines ideas from *Abstract Separation Logic*, *Variables as Resource* and Smallfoot
  - parser for Smallfoot example files
  - completely automatic verification
  - interactive proofs are possible as well
  - most features of Smallfoot are supported

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 may be used as a basis for a separation logic framework in HOL

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 $\exists s_1, s_2.\ (s = s_1 \circ s_2) \wedge s_1 \models P_1\ \wedge\ s_2 \models P_2$

Separation Combinator

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**
  $\forall x. \exists u_x.\ u_x \circ x = x$

Local Actions / Frame Rule

- partial correctness considered
- local reasoning essential

Frame Rule

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

- actions that respect the frame rule are called **local**
- just local actions will be considered in the following

Programs

- c for every local action c
- `p ; q`
- `p + q`
- `p*`
- `p || q`
- `with l do p`
- `l.p`

Notice that `skip` and `assume c` for intuitionistic conditions c are local actions.

Conditional execution and loops can be mimiced using non-determistic choice and `assume`.

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

Smallfoot II

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)]
```

HOL implementation of Smallfoot

- formalisation of *Abstract Separation Logic* (Calcagno, O'Hearn, Yang; LICS '07)
- first case study: a tool similar to Smallfoot
  - combines ideas from *Abstract Separation Logic*, *Variables as Resource* and Smallfoot
  - parser for Smallfoot example files
  - completely automatic verification
  - interactive proofs are possible as well
  - most features of Smallfoot are supported

Comparison mergesort.sf - split

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

```
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)]
```

Demo mergesort.sf

Demo of Smallfoot implementation.

Demo II

Demo of Smallfoot implementation.

Future Work

- extend the Smallfoot example to interactive verification / more interesting specifications
- modify the underlying abstract framework
- try other case studies like verification of assembler code