An Embedding of Abstract Separation Logic in HOL

Thomas Tuerk

ARG Lunch, 19th February 2008

I thank Mike Gordon, Matthew Parkinson, Magnus Myreen, Alexey Gotsman and Viktor Vafeiadis for discussions and advise.

Overview

Previous Work

- I deeply embedded the verification conditions generated by Smallfoot in HOL
- a decision procedure for these verification conditions was implemented

Example from `list.sf`

```
list_copy(p) [list(p)] {
  local t;
  t = p;
  q = NULL;
  while(t != NULL) [list(q) * lseg(p,t) * list(t)] {
    sq = q;
    q = new();
    q->tl = sq;
    t = t->tl;
  }
} [list(p) * list(q)]

t_2!=t * t!=x * t_2 |-> tl:t * listseg(tl; t, 0) * listseg(tl; x, t_2) |-
listseg(tl; x, t) * listseg(tl; t, 0)
```


Previous Work

- **However**: that's not separation logic yet, programs are missing
- **Problem**: there are a lot of different programming languages / separation logics around
  - standard heap, stack model
  - permissions
  - variable as resource
  - different ways of handling concurrency

Important Question

What is the essential part of separation logic?

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
- thus, I formalised abstract separation logic as described in *Local Action and Abstract Separation Logic* 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** $\forall x. \exists u_x.\ u_x \circ x = x$

Hoare Triples and Actions

- consider partial correctness
- an action is a function from states to either a special failure state $\top$ or a set of states
- $\emptyset$ used to model actions that diverge
- $\{P\}$ action $\{Q\}$ iff forall states s such that $s \models P$ the action does not fail and $t \models Q$ forall $t \in \text{action}(s)$

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

Best Local Action (bla)

- $a_1 \leq a_2$ iff $\forall P\ Q.\ \{P\}a_2\{Q\} \Rightarrow \{P\}a_1\{Q\}$
- the set of actions forms a lattice with this order
- the **best local action** bla(P, Q) is defined as the supremum of all local actions with
  - action is a local action
  - $\{P\}$action$\{Q\}$
- bla(P, Q) is local
- $\{P\}$bla$(P, Q)\{Q\}$
- if $\{P\}$action$\{Q\}$ and action local then action $\leq$ bla(P, Q)

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-deterministic choice and `assume`.

Traces

- a program is associated with a set of traces
- a trace is a sequence of
  - atomic actions
  - lock allocations and deallocations
  - checks that actions don't interfere
- given an environment η that assigns a **precise lock invariant** the semantics of a trace are given by
  - $[[c]]\eta = c$
  - $[[p; q]]\eta = [[p]]\eta; [[q]]\eta$
  - $[[check(c_1, c_2)]]\eta = \ldots$
  - $[[P(l)]]\eta = \text{bla}(\text{emp}, \eta(l))$
  - $[[V(l)]]\eta = \text{bla}(\eta(l), \text{emp})$
- the semantics of a program are defined as the supremum of the semantics of its traces

Inferences

$$\frac{\{P\}p\{Q\}}{\{P * R\}p\{Q * R\}} \qquad \frac{\{P\}p_1\{Q\} \quad \{Q\}p_2\{R\}}{\{P\}p_1; p_2\{R\}}$$

$$\frac{\{P\}p\{P\}}{\{P\}p * \{P\}} \qquad \frac{\{P\}p_1\{Q\} \quad \{P\}p_2\{Q\}}{\{P\}p_1 + p_2\{Q\}}$$

$$\frac{\{P_1\}p_1\{Q_1\} \quad \{P_2\}p_2\{Q_2\}}{\{P_1 * P_2\}p_1||p_2\{Q_1 * Q_2\}} \qquad \frac{\{P * \eta(I)\}p\{Q * \eta(I)\}}{\{P\}\text{with } I \text{ do } p\{Q\}}$$

Problems and Future Work

Problems

- semantics and verification are interleaved
- `assume` is not intuitive
- divergence and partial correctness used as a cheat

Future Work

- change the program semantics
- add procedures
- switch to C-like thread-semantics
- add tools to apply inferences

Work in HOL

- formalisation of abstract separation logic
- some work was done on lattices
- inference rules were proved correct