A Deep Embedding of a Decidable Fragment of Separation Logic in HOL

Thomas Tuerk Mike Gordon

ARG Lunch, 26th June 2006

Overview

Motivation

Basic Definitions

A Decision Procedure for Entailments

HOL embedding

Example

Conclusions and Future Work

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

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)]
```

Motivation

- we deeply embedded the fragment of separation logic used by Smallfoot in HOL
- a decision procedure for entailments has been implemented in HOL
- this formalisation may increase the trust in Smallfoot
- it may be used as a basis for non decidable fragments of separation logic and an interactive proof environment
- it may be extended to a HOL implementation of Smallfoot

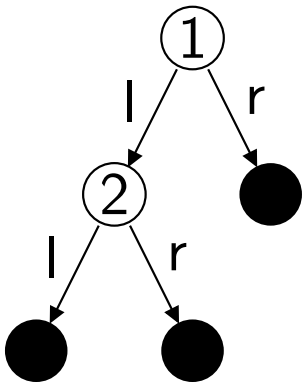
Pure Formulae

- an expression is either a constant or a variable
- nil is a special constant
- a stack is a variable assignment
- pure formulae are inductively defined by
  - true
  - $e_1 \doteq e_2$, $e_1 \not\doteq e_2$
  - $pf_1 \wedge pf_2$
- the semantics of pure formulae with respect to a stack are defined in the natural way

Spatial Formulae

- a heap is finite map $h : (\mathit{Values} \setminus \{\mathit{nil}\}) \xrightarrow{\mathit{fin}} \mathit{Fields} \xrightarrow{\mathit{fin}} \mathit{Values}$
- spatial formulae are inductively defined by
  - emp
  - $e \hookrightarrow [t_1 : e_1, \ldots, t_n : e_n]$
  - $\mathit{sf}_1 * \mathit{sf}_2$
  - $\mathsf{tree}((t_1, \ldots, t_n), es, e)$
- list segments and binary trees are defined as syntactic sugar
  - $\mathsf{bin\text{-}tree}(l, r, e) := \mathsf{tree}((l, r), \mathit{nil}, e)$
  - $\mathsf{ls}(tl, e_1, e_2) := \mathsf{tree}(tl, e_2, e_1)$

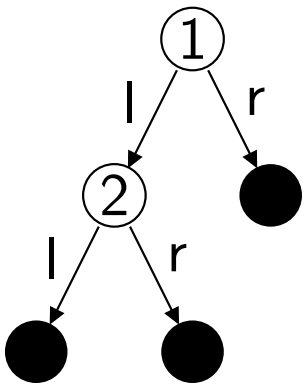
Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \rightarrow [l \rightarrow 2, r \rightarrow nil], 2 \rightarrow [l \rightarrow nil, r \rightarrow nil]]$$

$$\text{bin-tree}(l, r, x_1)$$

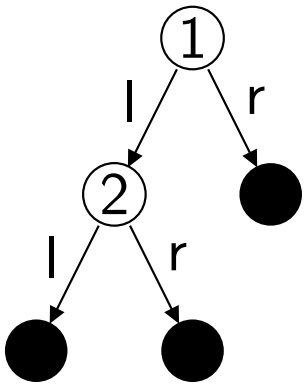
Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \rightarrow [l \rightarrow 2, r \rightarrow nil], 2 \rightarrow [l \rightarrow nil, r \rightarrow nil]]$$

$$\mathsf{tree}((l, r), \mathsf{nil}, x_1)$$

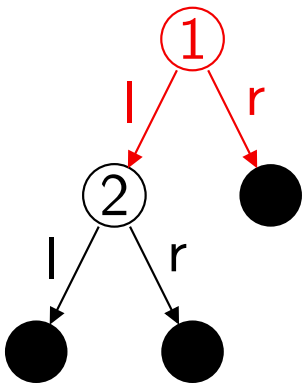
Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \rightarrow [l \rightarrow 2, r \rightarrow nil], 2 \rightarrow [l \rightarrow nil, r \rightarrow nil]]$$

$$\exists e_l, e_r.\ x_1 \hookrightarrow [l : e_l, r : e_r] * \mathsf{tree}((l, r), \mathsf{nil}, e_l) * \mathsf{tree}((l, r), \mathsf{nil}, e_r)$$

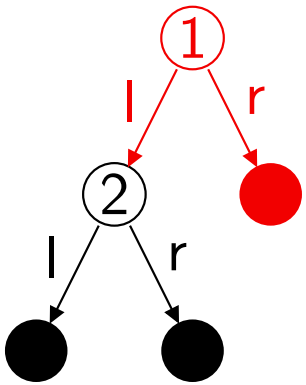
Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \rightarrow [l \rightarrow 2, r \rightarrow nil], 2 \rightarrow [l \rightarrow nil, r \rightarrow nil]]$$

$$x_1 \hookrightarrow [l : 2, r : \mathrm{nil}] * \mathrm{tree}((l, r), \mathrm{nil}, 2) * \mathrm{tree}((l, r), \mathrm{nil}, \mathrm{nil})$$

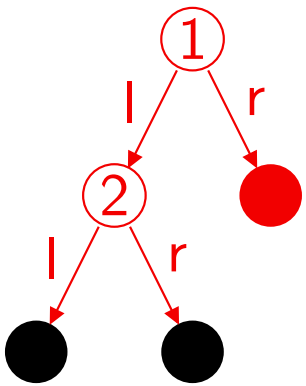
Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \rightarrow [l \rightarrow 2, r \rightarrow nil], 2 \rightarrow [l \rightarrow nil, r \rightarrow nil]]$$

$$x_1 \hookrightarrow [l : 2, r : \mathrm{nil}] * \mathrm{tree}((l, r), \mathrm{nil}, 2) * \mathrm{tree}((l, r), \mathrm{nil}, \mathrm{nil})$$

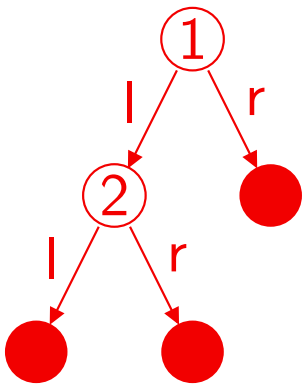
Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \rightarrow [l \rightarrow 2, r \rightarrow nil], 2 \rightarrow [l \rightarrow nil, r \rightarrow nil]]$$

$$x_1 \hookrightarrow [l : 2, r : \mathsf{nil}] * 2 \hookrightarrow [l : \mathsf{nil}, r : \mathsf{nil}] *$$
$$\mathsf{tree}((l, r), \mathsf{nil}, \mathsf{nil}) * \mathsf{tree}((l, r), \mathsf{nil}, \mathsf{nil}) * \mathsf{tree}((l, r), \mathsf{nil}, \mathsf{nil})$$

Spatial Formulae II



$$s(x_i) := i$$
$$h := [1 \longrightarrow [l \longrightarrow 2, r \longrightarrow nil], 2 \longrightarrow [l \longrightarrow nil, r \longrightarrow nil]]$$

$$x_1 \hookrightarrow [l : 2, r : \mathrm{nil}] * 2 \hookrightarrow [l : \mathrm{nil}, r : \mathrm{nil}] *$$
$$\mathrm{tree}((l, r), \mathrm{nil}, \mathrm{nil}) * \mathrm{tree}((l, r), \mathrm{nil}, \mathrm{nil}) * \mathrm{tree}((l, r), \mathrm{nil}, \mathrm{nil})$$

Entailments

- entailments are important for Smallfoot
- example from `list.sf`

$$x_0 \neq \mathsf{nil}, x_1 \neq \mathsf{nil}, x_2 \neq \mathsf{nil}, x_0 \neq x_3, x_0 \neq x_2, x_4 \neq x_5,$$
$$x_1 \neq x_3, x_1 \neq x_2, x_3 \neq x_2,$$
$$x_2 \hookrightarrow [hd : x_5, tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash$$
$$\mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil})$$

- entailments in this fragment of separation logic are decidable
- inferences can be easily combined to form a decision procedure
- inferences and decision procedure are presented in

  Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn.
  Symbolic Execution with Separation Logic.
  In K. Yi (Ed.): APLAS 2005, LNCS 3780, pp. 52-68, 2005.

RemoveTrivial

$$\textsc{RemoveTrivial-EQ-L}\ \frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{e \doteq e, \Pi, \Sigma \vdash \Pi', \Sigma'}$$

$$\textsc{RemoveTrivial-EQ-R}\ \frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \Sigma \vdash e \doteq e, \Pi', \Sigma'}$$

$$\textsc{RemoveTrivial-EmpTree-L}\ \frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \mathsf{tree}((t_1, \ldots, t_k), e, e), \Sigma \vdash \Pi', \Sigma'}$$

$$\textsc{RemoveTrivial-EmpTree-R}\ \frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, \Sigma \vdash \Pi', \mathsf{tree}((t_1, \ldots, t_k), e, e), \Sigma'}$$

Hypothesis

$$\textsc{Hypothesis}\ \frac{pf, \Pi, \Sigma \vdash \Pi', \Sigma'}{pf, \Pi, \Sigma \vdash pf, \Pi', \Sigma'}$$

Axiom / Inconsistent

AXIOM

$$\frac{}{\Pi \vdash}$$

INCONSISTENT-UNEQUAL

$$\frac{}{e \neq e, \Pi, \Sigma \vdash \Pi', \Sigma'}$$

INCONSISTENT-POINTSTO-NIL

$$\frac{}{\Pi, \mathsf{nil} \hookrightarrow [\ldots], \Sigma \vdash \Pi', \Sigma'}$$

Frame

$$\textsc{Frame-base}\ \frac{\Pi, \Sigma \vdash \Pi', \Sigma'}{\Pi, sf, \Sigma \vdash \Pi', sf, \Sigma'}$$

- problem: this is a real implication, information is lost
- thus, order of inference application matters
- example: $e \hookrightarrow [f : e_1], e \hookrightarrow [g : e_2] \vdash e \hookrightarrow [f : e_1]$
- solution: add additional informations to entailments

Frame II

FRAME-POINTS_TO

$$\frac{e, \eta, \pi, \Pi, \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, \Pi, e \hookrightarrow [t_1 : e_1, \ldots, t_n : e_n], \Sigma \vdash \Pi', e \hookrightarrow [t_1 : e_1, \ldots, t_m : e_m], \Sigma'} m \leq n$$

FRAME-TREE

$$\frac{\eta, (e, es), \pi, \Pi, \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, \Pi, \mathsf{tree}((t_1, \ldots, t_k), es, e), \Sigma \vdash \Pi', \mathsf{tree}((t_1, \ldots, t_k), es, e), \Sigma'}$$

- all used inferences are equivalences
- order of application does not matter
- continued applications of inferences will terminate

Substitution

$$\textsc{Substitution}\ \frac{\eta[e/x], \pi[e/x], \Pi[e/x], \Sigma[e/x] \ \vdash\ \Pi'[e/x], \Sigma'[e/x]}{\eta, x \doteq e, \pi, \Pi, \Sigma \ \vdash\ \Pi', \Sigma'}$$

NIL-NOT-LVAL

$$\frac{\eta, \pi, e \neq \mathsf{nil}, \Pi, e \hookrightarrow [\ldots], \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, \Pi, e \hookrightarrow [\ldots], \Sigma \vdash \Pi', \Sigma'}\ \textsc{nil-not-lval-pointsto}$$

$$\frac{\eta, \pi, e \neq \mathsf{nil}, e \neq es, \Pi, \mathsf{tree}\,(\ldots, es, e), \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, e \neq es, \Pi, \mathsf{tree}\,(\ldots, es, e), \Sigma \vdash \Pi', \Sigma'}\ \textsc{nil-not-lval-tree}$$

side condition

In order to prevent looping only new facts are added.

Partial

$$\textsc{Partial-pointsto-pointsto}\ \frac{\eta, \pi, e_1 \not\doteq e_2, \Pi, e_1 \hookrightarrow [\ldots], e_2 \hookrightarrow [\ldots], \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, \Pi, e_1 \hookrightarrow [\ldots], e_2 \hookrightarrow [\ldots], \Sigma \vdash \Pi', \Sigma'}$$

$$\textsc{Partial-pointsto-tree}\ \frac{\eta, \pi, e_1 \not\doteq e_2, e_2 \not\doteq e_3, \Pi, e_1 \hookrightarrow [\ldots], \mathsf{tree}(\ldots, e_3, e_2), \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, e_2 \not\doteq e_3, \Pi, e_1 \hookrightarrow [\ldots], \mathsf{tree}(\ldots, e_3, e_2), \Sigma \vdash \Pi', \Sigma'}$$

$\vdots$

side condition

In order to prevent looping only new facts are added.

Simple Unroll

UNROLL-RIGHT-LIST

$$\frac{e_1, \eta, \pi, e_1 \neq e_3, \Pi, \Sigma \vdash \Pi', \mathsf{ls}(tl, e_2, e_3), \Sigma'}{\eta, \pi, e_1 \neq e_3, \Pi, e_1 \hookrightarrow [tl : e_2, \ldots], \Sigma \vdash \Pi', \mathsf{ls}(tl, e_1, e_3), \Sigma'}$$

UNROLL-RIGHT-BINTREE

$$\frac{e, \eta, \pi, \Pi, \Sigma \vdash \Pi', \mathsf{bin\text{-}tree}(l, r, e_l), \mathsf{bin\text{-}tree}(l, r, e_r), \Sigma'}{\eta, \pi, \Pi, e \hookrightarrow [l : e_l, r : e_r, \ldots], \Sigma \vdash \Pi', \mathsf{bin\text{-}tree}(l, r, e), \Sigma'}$$

UNROLL-NILLIST

$$\frac{\eta, \pi, e \doteq \mathsf{nil}, \Pi, \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, \Pi, \mathsf{ls}(tl, \mathsf{nil}, e), \Sigma \vdash \Pi', \Sigma'}$$

UNROLL-PRECOND-LIST

$$\frac{e_1, \eta, \pi, e_1 \doteq e_2, \Pi, \Sigma \vdash \Pi', \Sigma'}{e_1, \eta, \pi, \Pi, \mathsf{ls}(tl, e_1, e_2), \Sigma \vdash \Pi', \Sigma'}$$

Unroll

$$\textsc{Unroll-list}\ \frac{\eta, \pi, e_1 \doteq e_2, \Pi, \Sigma \vdash \Pi', \Sigma' \qquad \forall x.\ \eta, \pi, e_1 \not\doteq e_2, e_2 \not\doteq x, \Pi, e_1 \hookrightarrow [\mathit{hd} : x], x \hookrightarrow [\mathit{hd} : e_2], \Sigma \vdash \Pi', \Sigma'}{\eta, \pi, \Pi, \mathsf{ls}(\mathit{tl}, e_1, e_2), \Sigma \vdash \Pi', \Sigma'}$$

- logic can not consider the content of a list
- therefore, two cases are sufficient
- no induction needed!
- similar inference exists for arbitrary trees
- however, in general useful for decision procedure

Append-list

$$\textsc{Append-list}\ \frac{\eta, (e_1, e_2), \pi, e_1 \neq e_3, \Pi, \Sigma \vdash \Pi', \mathsf{ls}(tl, e_2, e_3), \Sigma'}{\eta, \pi, e_1 \neq e_3, \Pi, \mathsf{ls}(tl, e_1, e_2), \Sigma \vdash \Pi', \mathsf{ls}(tl, e_1, e_3), \Sigma'}$$

- this inference holds under some complicated side condition
- it is preferable to unrolling lists
- its correctness proof uses unrolling of lists and simple unrolls and the frame inference

Decision Procedure

- these inferences can be easily combined to form a decision procedure for entailments
- apply inferences in arbitrary order as long as possible
- be careful with NIL-NOT-LVAL, Partial, Hypothesis to avoid looping
- iff the entailment could not be reduced to true, it is false
- remaining entailments are as simple, that a concrete counterexample can be easily constructed

HOL embedding

- deep embedding in HOL is straight forward except for trees
- trees are introduced considering their maximal depth
- equivalence with other recursive definition is formally proofed
- inferences are implemented as conversions
- the decision procedure is implemented as a conversion

HOL embedding II

	LOC
deep embedding and inferences	approx. 10 000
special versions of inferences for conversions	approx. 2 500
conversions and decision procedure	approx. 2 000

example from `list.sf`

$$x_0 \neq \mathsf{nil}, x_1 \neq \mathsf{nil}, x_2 \neq \mathsf{nil}, x_0 \neq x_3, x_0 \neq x_2, x_4 \neq x_5,$$
$$x_1 \neq x_3, x_1 \neq x_2, x_3 \neq x_2,$$
$$x_2 \hookrightarrow [hd : x_5, tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash$$
$$\mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil})$$

example from `list.sf`

$$\Pi,\ x_2 \hookrightarrow [hd : x_5, tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash \quad \mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil})$$

example inference application

$$\Pi, x_2 \hookrightarrow [hd : x_5, tl : x_3],$$
$$\mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash$$
$$\mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil})$$

example inference application

$$\Pi, x_2 \hookrightarrow [hd : x_5, tl : x_3],$$
$$\mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash$$
$$\mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil})$$

FRAME

$\Longleftrightarrow$

$$x_2, (x_3, \mathsf{nil}), \Pi, \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash$$
$$\mathsf{ls}(tl, x_1, x_2)$$

example inference application

$$\Pi, x_2 \hookrightarrow [hd : x_5, tl : x_3],\ \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}) \quad \vdash$$

FRAME

$\Longleftrightarrow$

$$x_2, (x_3, \mathsf{nil}), \Pi, \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2] \quad \vdash \quad \mathsf{ls}(tl, x_1, x_2)$$

APPEND-LIST

$\Longleftrightarrow$

$$x_2, (x_3, \mathsf{nil}), (x_1, x_0), \Pi, x_0 \hookrightarrow [tl, x_2] \quad \vdash \quad \mathsf{ls}(tl, x_0, x_2)$$

example inference application

$$\Pi, x_2 \hookrightarrow [hd : x_5, tl : x_3],\ \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}) \quad \vdash$$

FRAME

$$\Longleftrightarrow$$

$$x_2, (x_3, \mathsf{nil}), \Pi, \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_1, x_2) \quad \vdash$$

APPEND-LIST

$$\Longleftrightarrow$$

$$x_2, (x_3, \mathsf{nil}), (x_1, x_0), \Pi, x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_0, x_2) \quad \vdash$$

SIMPLE-UNROLL

$$\Longleftrightarrow$$

$$x_2, x_0, (x_3, \mathsf{nil}), (x_1, x_0), \Pi\ \mathsf{ls}(tl, x_2, x_2) \quad \vdash$$

example inference application

$$\Pi, x_2 \hookrightarrow [hd : x_5, tl : x_3],\ \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2],\ \mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}) \vdash$$

FRAME $\Longleftrightarrow$

$$x_2, (x_3, \mathsf{nil}), \Pi, \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2],\ \mathsf{ls}(tl, x_1, x_2) \vdash$$

APPEND-LIST $\Longleftrightarrow$

$$x_2, (x_3, \mathsf{nil}), (x_1, x_0), \Pi, x_0 \hookrightarrow [tl, x_2],\ \mathsf{ls}(tl, x_0, x_2) \vdash$$

SIMPLE-UNROLL $\Longleftrightarrow$

$$x_2, x_0, (x_3, \mathsf{nil}), (x_1, x_0), \Pi,\ \mathsf{ls}(tl, x_2, x_2) \vdash$$

REMOVE-TRIVIAL $\Longleftrightarrow$

$$x_2, x_0, (x_3, \mathsf{nil}), (x_1, x_0), \Pi \vdash$$

example inference application

$$\Pi, x_2 \hookrightarrow [hd : x_5, tl : x_3],\ \mathsf{ls}(tl, x_3, \mathsf{nil}), \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_1, x_2), x_2 \hookrightarrow [tl : x_3], \mathsf{ls}(tl, x_3, \mathsf{nil}) \quad \vdash \quad \textsc{Frame} \quad \Longleftrightarrow$$

$$x_2, (x_3, \mathsf{nil}), \Pi, \mathsf{ls}(tl, x_1, x_0), x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_1, x_2) \quad \vdash \quad \textsc{Append-List} \quad \Longleftrightarrow$$

$$x_2, (x_3, \mathsf{nil}), (x_1, x_0), \Pi, x_0 \hookrightarrow [tl, x_2]\ \mathsf{ls}(tl, x_0, x_2) \quad \vdash \quad \textsc{Simple-Unroll} \quad \Longleftrightarrow$$

$$x_2, x_0, (x_3, \mathsf{nil}), (x_1, x_0), \Pi\ \mathsf{ls}(tl, x_2, x_2) \quad \vdash \quad \textsc{Remove-Trivial} \quad \Longleftrightarrow$$

$$x_2, x_0, (x_3, \mathsf{nil}), (x_1, x_0), \Pi \quad \vdash \quad \textsc{Axiom} \quad \Longleftrightarrow$$

$$\top$$

example HOL

```
val t = ``LIST_DS_ENTAILS ([],[])
  ([pf_unequal (dse_var 0) dse_nil;
    pf_unequal (dse_var 1) dse_nil;
    pf_unequal (dse_var 2) dse_nil;
    pf_unequal (dse_var 0) (dse_var 3);
    pf_unequal (dse_var 0) (dse_var 2);
    pf_unequal (dse_var 4) (dse_var 5);
    pf_unequal (dse_var 1) (dse_var 3);
    pf_unequal (dse_var 1) (dse_var 2);
    pf_unequal (dse_var 3) (dse_var 2)],
   [sf_points_to (dse_var 2) [("hd",dse_var 5); ("tl", dse_var 3)];
    sf_ls "tl" (dse_var 3) dse_nil;
    sf_ls "tl" (dse_var 1) (dse_var 0);
    sf_points_to (dse_var 0) [("tl", (dse_var 2))]])

   ([],
    [sf_ls "tl" (dse_var 1) (dse_var 2);
     sf_points_to (dse_var 2) [("tl", dse_var 3)];
     sf_ls "tl" (dse_var 3) dse_nil])``;
```

example HOL II

```
val thm1 =
(ds_inference_FRAME___CONV THENC
 ds_inference_APPEND_LIST___CONV THENC
 ds_inference_SIMPLE_UNROLL___CONV THENC
 ds_inference_REMOVE_TRIVIAL___CONV THENC
 ds_inference_AXIOM___CONV) t;

val thm2 = ds_DECIDE_CONV t;
```

Conclusions

- there is a deep embedding of the decidable fragment of separation logic used by Smallfoot
- all inferences used by Smallfoot have been verified using HOL
- a decision procedure for entailments has been implemented

Future Work

- add symbolic execution to build a Smallfoot implementation in HOL
- extend the logic
- try interactive proofs for more complicated fragments of separation logic