

Smallfoot: Modular Automatic Assertion Checking with Separation Logic

Cristiano Calcagno
Imperial College London

(with Josh Berdine and Peter O'Hearn)



Smallfoot v0.1

<http://www.dcs.qmul.ac.uk/research/logic/theory/projects/smallfoot/>

Implementation: Berdine, Calcagno.

Design and Theory: Berdine, Calcagno, O'Hearn.

Testing and cool examples: Parkinson.

Thanks to: Richard Bornat, Steve Brookes, Dino Distefano, Ivana Mijajlovic, Uday Reddy, John Reynolds, Hongseok Yang.

Separation Logic

Combination of two important aspects:

- Fault-avoiding interpretation of triples

$$\{P\} C \{Q\}$$

- Separation and frame rule

$$\frac{\{P\} C \{Q\}}{\{P * R\} C \{Q * R\}}$$

$P * Q$: expressing separation nicely

$P - * Q$: “structural” assume/guarantee



Automatic Reasoning

Verification tool - standard structure with Hoare Logic

- User provides annotations (formulae) for pre- and post-conditions of procedures and loop invariants
- Verification condition generator
- Validity checker (for weakest preconditions)

Example

- Annotated program: $\{P\} C \{Q\}$
- Verification condition: $P \Rightarrow wp(C, Q)$
- Check: $\forall s, h. s, h \models P \Rightarrow wp(C, Q)$



Validity Checker

How do we check validity?

- Use interactive theorem proving (Isabelle, COQ, PVS,...)
- Completely automatic decision procedure

Want automatic tool

- Safety properties (pointer errors, memory leaks) vs. full functional correctness
- Need decidable fragment (full SL is undecidable)



The Theory Behind Smallfoot



Towards a Fragment

Weakest preconditions

- $\{\forall x. (x \mapsto 0 \text{ -* } P)\} x := \text{new}(); \{P\}$
- \forall and -* , forget decidability

Going forwards

- Start from precondition
 $\{P_1\} C_1 \{P_2\} C_2 \{P_3\} C_3 \{P_4\} \dots$
- Strongest postcondition: what is it?
 $\{\text{emp}\} x \rightarrow \text{tl} := y \{???\}$
- Need safe precondition



Symbolic Execution

From precondition in explicit safe form

$A,$	$x := E$	\rightarrow	$x = E[x'/x] \wedge A[x'/x]$
$A * E \mapsto F,$	$x := [E]$	\rightarrow	$x = F[x'/x] \wedge (A * E \mapsto F)[x'/x]$
$A * E \mapsto F,$	$[E] := G$	\rightarrow	$A * E \mapsto G$
$A,$	$x := \text{new}()$	\rightarrow	$A[x'/x] * x \mapsto F$
$A * E \mapsto F,$	$\text{dispose } E$	\rightarrow	A

No need for quantifiers



The Fragment

Assertions A of special form: “symbolic heaps”

$A ::= (P \wedge \dots \wedge P) ; (S * \dots * S)$

$E ::= x \mid \text{nil} \mid E \wedge E$

$P ::= E = E \mid E \neq E$

$S ::= E \mapsto E_1 \dots E_n \mid \text{tree}(E) \mid \text{ls}(E, E) \mid \text{dlseg}(E, E, E, E) \mid$
 $\text{xlseg}(E, E, E, E)$

$\text{ls}(E, F) \Leftrightarrow$ if $E = F$ then emp else $\exists x. E \mapsto x * \text{ls}(x, F)$

$\text{tree}(E) \Leftrightarrow$ if $E = \text{nil}$ then emp else $\exists x, y. E \mapsto x, y * \text{tree}(x) * \text{tree}(y)$



Pure Theory and Spatial Theory

Nelson-Oppen: combine pure theories

Here: pure part P and heap part S

Intuition: $A_1 * A_2$ expresses inequalities and equalities at boundary of data structures

Constraint generation:

From $S_1 * \dots * S_n$ to constraints of form:

$$(E_1 \neq F_1 \wedge \dots \wedge E_k \neq F_k) \Rightarrow P$$



Constraints

Generation example

From $Is(x,y) * Is(z,w)$ generate:

$$x \neq y \Rightarrow x \neq nil$$

$$z \neq w \Rightarrow z \neq nil$$

$$(x \neq y \wedge z \neq w) \Rightarrow x \neq z$$

Constraint solving

Only interested in pure consequences

$$P_1 \wedge \dots \wedge P_k$$



The Need for Induction

For each (hardwired) inductive predicate:

- rules axiomatizing consequences of induction, e.g.:

$$\text{Is}(E_1, E_2) * \text{Is}(E_2, \text{nil}) \vdash \text{Is}(E_1, \text{nil})$$

but not: $\text{Is}(E_1, E_2) * \text{Is}(E_2, E_3) \vdash \text{Is}(E_1, E_3)$

- formulated without needing to enumerate potential induction hypotheses
- compatible with a terminating proof theory

For trees and list segments:

proof theory complete (entailment decidable)



Entailments Between Symbolic Heaps

Terminating proof theory for entailments $A \vdash A'$

Normalization Rules

- recognize inconsistency
- get rid of equalities via substitution
- make derivable inequalities syntactically explicit, e.g.:

$$\frac{E_1 \neq E_2 \wedge E_2 \neq F_2 \wedge A * E_1 \mapsto F_1 * \text{ls}(E_2, F_2) \vdash B}{E_2 \neq F_2 \wedge A * E_1 \mapsto F_1 * \text{ls}(E_2, F_2) \vdash B} \quad E_1 \neq E_2 \notin A$$

- and perform case analysis using a form of excluded middle:

$$\frac{E_1 = E_2 \wedge A \vdash B \quad E_1 \neq E_2 \wedge A \vdash B}{A \vdash B} \quad \begin{array}{l} E_1 \neq E_2 \\ E_1 = E_2, E_1 \neq E_2 \notin A \\ \text{fv}(E_1, E_2) \subseteq \text{fv}(A, B) \end{array}$$



Entailments Between Symbolic Heaps

Terminating proof theory for entailments $A \vdash A'$

Subtraction Rules

$$\frac{E_1 \neq E_3 \wedge A * E_1 \mapsto E_2 \vdash B * E_1 \mapsto E_2 * \text{ls}(E_2, E_3)}{E_1 \neq E_3 \wedge A * E_1 \mapsto E_2 \vdash B * \text{ls}(E_1, E_3)} \quad E_1 \mapsto E_2 \notin B$$

$$\frac{A * \text{ls}(E_1, E_2) \vdash B * \text{ls}(E_1, E_2) * \text{ls}(E_2, \text{nil})}{A * \text{ls}(E_1, E_2) \vdash B * \text{ls}(E_1, \text{nil})} \quad \text{ls}(E_1, E_2) \notin B$$



More Than Just Implications

Recall: fault-avoiding interpretation and frame

Fault-avoiding: forward execution

Frame: how do we apply it?

$$\frac{\{P\} C \{Q\}}{\{P * R\} C \{Q * R\}}$$

Generalized implication: frame inference

$$A \vdash A' * F$$

Procedure calls:

$$\frac{A \vdash A' * F}{A, [A'] \text{ jsr } [B'] \rightarrow B' * F}$$



Example: tree_deallocate

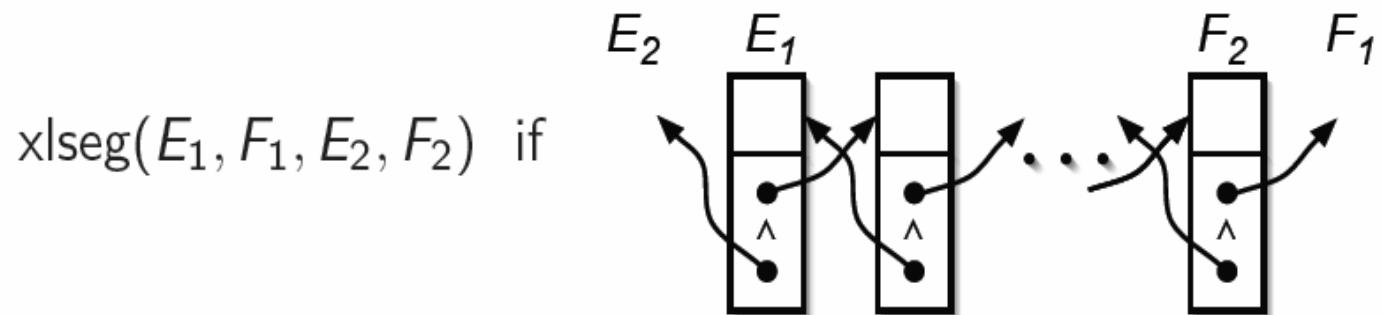
```
tree_deallocate(t) {           // tree(t)
  local i,j;
  if(t == NULL) {}
  else {
    i = t→l;
    j = t→r;                   // t ↦ i,j * tree(j) * tree(i)
    tree_deallocate(i);       // t ↦ i,j * tree(j) * emp
    tree_deallocate(j);       // t ↦ i,j
    dispose t;                // emp
  }
}                               // emp
```

```
{tree(t)}
tree_deallocate(t)
{emp}
```

$$\frac{\{P\} C \{Q\}}{\{P * R\} C \{Q * R\}}$$


Dirty Features: Address Arithmetic

- Reachability is irrelevant
- Simple address arithmetic



- In practice, procedure preconditions often not closed under reachability (point into frame)



Smallfoot Today

```
emacs@Dell
File Edit Options Buffers Tools Help

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)]

--- parallel_mergesort.sf (Fundamental)--L42--Bot-----
File "parallel_mergesort.sf", line 42, characters 4-39:
Intermediate State:
[p_2==p * 0!=p * listseg(tl; p_2, 0) * listseg(tl; q, 0)]
fcall({},emp, p_3==q * p_4==p);
fcall({p1, q1},listseg(tl; p_3, 0) * listseg(tl; p_4, 0),
      listseg(tl; q1, 0) * listseg(tl; p1, 0));
fcall({},emp, p_5==p1 * q_3==q1);
fcall({p_5, q_3, r},listseg(tl; p_5, 0) * listseg(tl; q_3, 0),
      listseg(tl; r, 0));
[listseg(tl; r, 0)]

File "parallel_mergesort.sf", line 42, characters 4-39:
Intermediate State:
[p_2==p * p_3==q * p_4==p * 0!=p * p!=q * listseg(tl; p, 0) * listseg(tl; q, 0)]
fcall({p1, q1},listseg(tl; p_3, 0) * listseg(tl; p_4, 0),
      listseg(tl; q1, 0) * listseg(tl; p1, 0));
--- ** *compilation* (Compilation:exit [0])--L456--89%-----
```



V0.1 Features

- Implementation: under 4000 lines of (dense) OCaml
- Handles C-like toy language
- Functions: reference and value parameters
- Hard-coded inductive predicates for: list segments, trees, doubly-linked and xor-linked lists
- Simple memory model: fixed collection of field names
- Concurrency: conditional critical regions, parallel function calls



Smallfoot Tomorrow

```
emacs@Dell
File Edit Options Buffers Tools Help

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)]

--- parallel_mergesort.sf (Fundamental)--L42--Bot-----
File "parallel_mergesort.sf", line 42, characters 4-39:
Intermediate State:
[p_2==p * 0!=p * listseg(tl; p_2, 0) * listseg(tl; q, 0)]
fcall({},emp, p_3==q * p_4==p);
fcall({p1, q1},listseg(tl; p_3, 0) * listseg(tl; p_4, 0),
      listseg(tl; q1, 0) * listseg(tl; p1, 0));
fcall({},emp, p_5==p1 * q_3==q1);
fcall({p_5, q_3, r},listseg(tl; p_5, 0) * listseg(tl; q_3, 0),
      listseg(tl; r, 0));
[listseg(tl; r, 0)]

File "parallel_mergesort.sf", line 42, characters 4-39:
Intermediate State:
[p_2==p * p_3==q * p_4==p * 0!=p * p!=q * listseg(tl; p, 0) * listseg(tl; q, 0)]
fcall({p1, q1},listseg(tl; p_3, 0) * listseg(tl; p_4, 0),
      listseg(tl; q1, 0) * listseg(tl; p1, 0));
--- ** *compilation* (Compilation:exit [0])--L456--89%-----
```



Where it is going

- Existentials and logical variables
- User-defined inductive definitions
- Whole C language
- Inferring loop invariants
- Inferring procedure specs
- Permissions
- Variables as resources



Existentials

Existential prenex form

Can do e.g. circular lists:

$$\exists y. \text{ls}(x,y) * \text{ls}(y,x)$$

Preconditions: for free

Postconditions: general implication

- Guess frames
- Guess instantiation of existentials

Logical variables in specs

$$\{x \mapsto _y\} f(x) \{ \text{emp} \wedge z = _y \}$$



User-defined predicates

Simple grammar for definitions:

$$p(x) \Leftrightarrow A$$

Example

$$\text{odd}(E,F) \Leftrightarrow \exists x. E \mapsto x * \text{even}(x,F)$$

$$\text{even}(E,F) \Leftrightarrow \text{if } E=F \text{ then emp else } \exists x. E \mapsto x * \text{odd}(x,F)$$

Heuristics for unrolling

Add a drop of arithmetic: red-black trees

