Ownership, Pointer Arithmetic and Memory Separation

Romain Bardou
INRIA Saclay, France

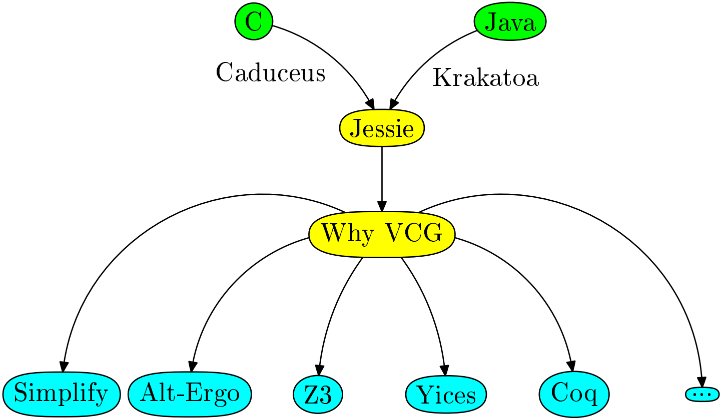
FTfJP2008

Jessie (1/2)

Context: deductive verification

INRIA Saclay: the Why platform to verify C or Java programs

Jessie (2/2)


C
Java
Caduceus
Krakatoa
Jessie
Why VCG
Simplify
Alt-Ergo
Z3
Yices
Coq
...

Ownership and Invariants (1/2)

Examples of invariants:

- $x \neq 0$
- $t.size = length(t.data)$
- Tree t is a search tree
- Tree t is balanced

Ownership and Invariants (2/2)

Existing systems:

- Spec# / Boogie
- Universes type system
- Capabilities
- ...

And in Jessie?

Problems

Memory model of Jessie:

- Pointer arithmetic
  - Used to encode *arrays*
  - A single pointer may be shifted to access several others
- Memory separation
  - Memory is *split* into several maps from pointers to values
  - Simplifies pointer aliasing problems
  - Global properties of the ownership system are harder to express

Our Approach

- Provide a small *core language* formalizing ownership and invariants
  - Captures the core ideas of ownership
  - Simple formalization usable in proofs
  - Easy to extend
    - Pointer arithmetic
- Express the *global properties* of the ownership system, *in the logic*, using *assumptions*

Syntax

e	$::=$	v	Values
	$\mid$	x	Variables
	$\mid$	$\textbf{let}\ x = e\ \textbf{in}\ e$	Binding
	$\mid$	$e; e$	Sequence
	$\mid$	$\textbf{while}\ e\ \textbf{do}\ e$	Turing-completion
	$\mid$	$\textbf{if}\ e\ \textbf{then}\ e\ \textbf{else}\ e$	Test
	$\mid$	$\textbf{new}\ \langle e; l; \mathbf{r} \rangle$	Allocation
	$\mid$	$!e$	Dereferencing
	$\mid$	$e := e$	Assignment
	$\mid$	$\textbf{pack}\ e$	Packing
	$\mid$	$\textbf{unpack}\ e$	Unpacking

Semantics (1/4)

Allocation (p fresh in Γ):

$$\Gamma; \textbf{new}\ \langle v; l; \mathbf{r} \rangle \quad \longrightarrow \quad \Gamma, p = \langle v; l; \mathbf{r} \rangle^{\circ}; p$$

Assignment:

$$\Gamma, p = \langle v_1; l; \mathbf{r} \rangle^{\circ}; p := v_2 \quad \longrightarrow \quad \Gamma, p = \langle v_2; l; \mathbf{r} \rangle^{\circ}; \mathrm{unit}$$

Semantics (2/4)

Packing (only if I holds in Γ):

$$\Gamma, \left(\begin{array}{l} p = \langle v; I; p_1 \cdots p_n \rangle^{\circ} \\ p_1 = R_1^{\times}, \cdots, p_n = R_n^{\times} \end{array} \right); \textbf{pack } p$$
$$\longrightarrow$$
$$\Gamma, \left(\begin{array}{l} p = \langle v; I; p_1 \cdots p_n \rangle^{\times} \\ p_1 = R_1^{\otimes}, \cdots, p_n = R_n^{\otimes} \end{array} \right); \text{unit}$$

Semantics (3/4)

Unpacking:

$$\Gamma, \left(\begin{array}{l} p = \langle v; l; p_1 \cdots p_n \rangle^{\times} \\ p_1 = R_1^{\otimes}, \cdots, p_n = R_n^{\otimes} \end{array} \right); \textbf{unpack}\ p$$

$$\longrightarrow$$

$$\Gamma, \left(\begin{array}{l} p = \langle v; l; p_1 \cdots p_n \rangle^{\circ} \\ p_1 = R_1^{\times}, \cdots, p_n = R_n^{\times} \end{array} \right); \mathsf{unit}$$

Semantics (4/4)

Assignment can only be done on open (unpacked) pointers:

$$\Gamma, p = \langle v_1; l; \mathbf{r} \rangle^\circ; p := v_2 \quad \longrightarrow \quad \Gamma, p = \langle v_2; l; \mathbf{r} \rangle^\circ; \mathrm{unit}$$

This ensures that invariants are not broken.

Example (1/2)

[Müller, challenges in Java program verification]

Let t be an array of integer of size n.

```
int i, j, count = 0;
for (i=0; i < t.length; i++)
  if (t[i] > 0) count++;
int u[] = new int[count];
for (i=0, j=0; i < n; i++)
  if (t[i] > 0) u[j++] = t[i];
```

This copies the positive elements of t in the new array u.

Problem: access `u[j++]` inside array bounds?

Example (2/2)

$$
\begin{array}{l}
\textbf{let } i = \textbf{new } \langle 0; \mathrm{true}; \emptyset \rangle \textbf{ in} \\
\textbf{let } j = \textbf{new } \langle 0; \mathrm{true}; \emptyset \rangle \textbf{ in} \\
\textbf{let } count = \textbf{new } \langle 0; (\lambda p.!p = Card\{i \mid t[i] > 0\}); t[0..n] \rangle \textbf{ in} \\
\textbf{while } !i \leq n \textbf{ do} \\
\quad (\textbf{if } !t[!i] > 0 \textbf{ then } count :=!count + 1; \\
\quad i :=!i + 1); \\
\textbf{pack } count; \\
\textbf{let } u = \textbf{new } \langle 0; \mathrm{true}; \emptyset \rangle [!count] \textbf{ in} \\
i := 0; \\
\textbf{while } !i \leq n \textbf{ do} \\
\quad (\textbf{if } !t[!i] > 0 \textbf{ then } (u[!j] :=!t[!i]; j :=!j + 1)) \\
\quad i :=!i + 1)
\end{array}
$$

Pointer Shifting

We axiomatize pointer shifting $\oplus$:

$$p \oplus i = p \iff i = 0$$
$$(p \oplus i) \oplus j = p \oplus (i + j)$$

Pointers do *not* have to be all related together by $\oplus$.

$\cdots$	$p \oplus -1$	p	$p \oplus 1$	$p \oplus 2$	$\cdots$

Extending Allocation

$$\textbf{new } \lambda o.\langle v; l(o); \mathbf{r}(o)\rangle[n]$$

allocates the following fresh pointers:

$$p \oplus o = \langle v; l(o); \mathbf{r}(o)\rangle$$

where $o \in \{0, \cdots, n-1\}$.

All these pointers are *known to be related* by $\oplus$.

Examples (1/3)

An array of positive integers:

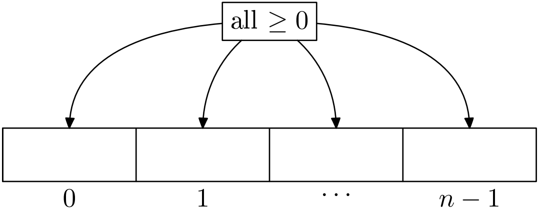
$$\textbf{new } \lambda o.\langle 0; (\lambda p.!p \geq 0); \emptyset\rangle[n]$$

≥ 0	≥ 0	$\cdots$	≥ 0
0	1	$\cdots$	$n-1$

Examples (2/3)

A pointer on an array of positive integers:

$$\begin{array}{l}\textbf{let } p = \textbf{new } \lambda o.\langle 0; \mathsf{true}; \emptyset\rangle[n] \textbf{ in} \\ \textbf{new } \langle 0; (\forall o,\ !(!p \oplus o) \geq 0); p \oplus [0..(n-1)]\rangle\end{array}$$



Examples (3/3)

$$
\begin{array}{l}
\textbf{let } i = \textbf{new } \langle 0; \mathrm{true}; \emptyset \rangle \textbf{ in} \\
\textbf{let } j = \textbf{new } \langle 0; \mathrm{true}; \emptyset \rangle \textbf{ in} \\
\textbf{let } \mathit{count} = \textbf{new } \langle 0; (\lambda p. !p = \mathit{Card}\{i \mid t \oplus i > 0\}); t \oplus [0..n] \rangle \textbf{ in} \\
\textbf{while } !i \leq n \textbf{ do} \\
\quad (\textbf{if } !t \oplus !i > 0 \textbf{ then } \mathit{count} := !\mathit{count} + 1; \\
\quad i := !i + 1); \\
\textbf{pack } \mathit{count}; \\
\textbf{let } u = \textbf{new } \lambda o. \langle 0; \mathrm{true}; \emptyset \rangle [!\mathit{count}] \textbf{ in} \\
i := 0; \\
\textbf{while } !i \leq n \textbf{ do} \\
\quad (\textbf{if } !t \oplus !i > 0 \textbf{ then } (u \oplus !j := !t \oplus !i; j := !j + 1)) \\
\quad i := !i + 1)
\end{array}
$$

With One Heap (1/3)

Global ownership properties:

- p is closed $\implies$ $Inv(p)$
- p is closed $\implies$ reps of p are owned
- Owner of p is unique

Axioms?

Depends on the heap.

In Spec# / Boogie: *IsHeap*

$$\forall h.\ IsHeap(h) \implies Global\_ownership\_properties(h)$$

With One Heap (2/3)

Another point of view: assumptions

The following code:

$x := !y$

becomes, at the Boogie or Why level:

assume $Global\_ownership\_properties(h)$
$h := store(h, x, select(h, y))$
assume $Global\_ownership\_properties(h)$

With One Heap (3/3)

Suppose a free variable x.

$$\textbf{let } y = \textbf{new } \langle 0; (\lambda p.!p > !x); x\rangle \textbf{ in } y := !x + 1; \textbf{pack } y; y$$

Post-condition: $!y > !x$

Resulting proof obligation (simplified):

$$\left.\begin{array}{l} \forall h, x, y \\ \mathit{IsHeap}(h) \\ \mathit{closed}(h, y) \end{array}\right\} \Longrightarrow \mathit{select}(h, y) > \mathit{select}(h, x)$$

With Multiple Memories

The heap h is split into several maps from pointers to values.

$$\left.\begin{array}{l} \forall h_x, h_y, x, y \\ \mathit{IsHeap}(???) \\ \mathit{closed}(h_y, y) \end{array}\right\} \Longrightarrow \mathit{select}(h_y, y) > \mathit{select}(h_x, x)$$

Linking Memories (1/2)

Axiom:

$$\left.\begin{array}{l} \forall h, h_x, h_y. \\ linked(h, h_x) \\ linked(h, h_y) \\ IsHeap(h) \end{array}\right\} \Longrightarrow Global\_ownership\_properties(h, h_x, h_y)$$

Proof obligation:

$$\left.\begin{array}{l} \forall h, h_x, h_y, x, y \\ linked(h, h_x) \\ linked(h, h_y) \\ IsHeap(h) \\ closed(h_y, y) \end{array}\right\} \Longrightarrow select(h_y, y) > select(h_x, x)$$

This is inconsistent!

Linking Memories (2/2)

Solution: use the assumption point of view.

...some code...
assume $\mathit{Global\_ownership\_properties}(\mathit{Current\_heap}_x, \mathit{Current\_heap}_y)$
...some code...

Proof obligation:

$$\left.\begin{array}{l} \forall h_x, h_y, x, y \\ \mathit{Global\_ownership\_properties}(h_x, h_y) \\ \mathit{closed}(h_y, y) \end{array}\right\} \Rightarrow \mathit{select}(h_y, y) > \mathit{select}(h_x, x)$$

Example

$\ldots$

pack *count*;

$\ldots$

$\textbf{while}\ !i \leq n\ \textbf{do}$
$\quad (\textbf{if}\ !t \oplus !i > 0\ \textbf{then}\ (u \oplus !j := !t \oplus !i; j := !j + 1))$
$\quad i := !i + 1)$

Thanks to memory separation:

- Contents of array t is not modified: trivial

Thanks to the invariant system:

- All accesses to array u are valid: immediate consequence of $closed(count)$ and the assumed global ownership properties

Conclusion

The power of the ownership system of Spec#...

- captured in a small core language,
- implemented with pointer arithmetic,
- and memory separation.

Other possible extensions such as classes (implemented in Jessie).

Could be used in ESC/Java, ...