Ownership, Pointer Arithmetic, and Memory Separation

Romain Bardou
INRIA Saclay, France

FTfJP2008
Introduction

Jessie
Ownership and Invariants
Problems
Our Approach

Core Language

Pointer Arithmetic

Memory Separation

Conclusion
Context: deductive verification

INRIA Saclay: the Why platform to verify C or Java programs
Simplify
Alt-Ergo
Z3
Yices
Coq
...

Ownership, Pointer Arithmetic, and Memory Separation
Introduction
Jessie
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*
Introduction

Core Language
  Syntax
  Semantics
  Example

Pointer Arithmetic

Memory Separation

Conclusion
Syntax

\[ e ::= v \quad \text{Values} \]
\[ \quad x \quad \text{Variables} \]
\[ \quad \textbf{let} \ x = e \ \textbf{in} \ e \quad \text{Binding} \]
\[ \quad e; e \quad \text{Sequence} \]
\[ \quad \textbf{while} \ e \ \textbf{do} \ e \quad \text{Turing-completion} \]
\[ \quad \textbf{if} \ e \ \textbf{then} \ e \ \textbf{else} \ e \quad \text{Test} \]
\[ \quad \textbf{new} \ <e; l; r> \quad \text{Allocation} \]
\[ \quad !e \quad \text{Dereferencing} \]
\[ \quad e ::= e \quad \text{Assignment} \]
\[ \quad \textbf{pack} \ e \quad \text{Packing} \]
\[ \quad \textbf{unpack} \ e \quad \text{Unpacking} \]
Semantics (1/4)

Allocation ($p$ fresh in $\Gamma$):

\[
\Gamma; \text{new} \langle v; l; r \rangle \rightarrow \Gamma, p = \langle v; l; r \rangle^\circ; \ p
\]

Assignment:

\[
\Gamma, p = \langle v_1; l; r \rangle^\circ; \ p := v_2 \rightarrow \Gamma, p = \langle v_2; l; r \rangle^\circ; \text{unit}
\]
Packing (only if \( I \) holds in \( \Gamma \)):

\[
\Gamma, \left( p = \langle v; I; p_1 \cdots p_n \rangle^\circ \right) \; \text{pack} \; p
\]

\[
p_1 = R_1^\times, \ldots, p_n = R_n^\times
\]

\[
\Gamma, \left( p = \langle v; I; p_1 \cdots p_n \rangle^\times \right) \; \text{unit}
\]

\[
p_1 = R_1^\otimes, \ldots, p_n = R_n^\otimes
\]
Unpacking:

\[
\Gamma, \left( \begin{array}{c}
p = \langle v; I; p_1 \cdots p_n \rangle^\times \\
p_1 = R_1^\otimes, \ldots, p_n = R_n^\otimes
\end{array} \right); \text{unpack } p
\]

\[
\Gamma, \left( \begin{array}{c}
p = \langle v; I; p_1 \cdots p_n \rangle^\circ \\
p_1 = R_1^\times, \ldots, p_n = R_n^\times
\end{array} \right); \text{unit}
\]
Assignment can only be done on open (unpacked) pointers:

\[ \Gamma, p = \langle v_1; l; r \rangle^\circ; \quad p := v_2 \quad \rightarrow \quad \Gamma, p = \langle v_2; l; r \rangle^\circ; \text{unit} \]

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 $n$.

```java
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)

```
let i = new ⟨0; true; ∅⟩ in
let j = new ⟨0; true; ∅⟩ in
let count = new ⟨0; (λp.!p = Card{ i | t[i] > 0}); t[0..n]⟩ in
while !i ≤ n do
  (if !t[!i] > 0 then count :=!count + 1;
   i :=!i + 1);
pack count;
let u = new ⟨0; true; ∅⟩[!count] in
i := 0;
while !i ≤ n do
  (if !t[!i] > 0 then (u[!j] :=!t[!i]; j :=!j + 1))
  i :=!i + 1)
```
Introduction

Core Language

Pointer Arithmetic
  Pointer Shifting
  Extending Allocation
  Examples

Memory Separation

Conclusion
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 $\oplus$. 

\[\begin{array}{cccc}
\cdots & p \oplus -1 & p & p \oplus 1 & p \oplus 2 & \cdots
\end{array}\]
Extending Allocation

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

allocates the following fresh pointers:

\[ p \oplus o = \langle v; l(o); r(o) \rangle \]

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:

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

\[
\begin{array}{cccc}
\geq 0 & \geq 0 & \cdots & \geq 0 \\
0 & 1 & \cdots & n - 1 \\
\end{array}
\]
A pointer on an array of positive integers:

```plaintext
let p = new λo.⟨0; true; ∅⟩[n] in
new ⟨0; (∀o, !(p ⊕ o) ≥ 0); p ⊕ [0..(n − 1)]⟩
```

![Diagram showing an array with elements from 0 to n-1 and a condition for all elements being greater than or equal to 0.](image)
let \( i = \text{new} \langle 0; \text{true}; \emptyset \rangle \) in
let \( j = \text{new} \langle 0; \text{true}; \emptyset \rangle \) in
let \( \text{count} = \text{new} \langle 0; (\lambda p. \text{!}p = \text{Card}\{ i \mid t \oplus i > 0 \}); t \oplus [0..n] \rangle \) in
while \( \text{!}i \leq n \) do
  (if \( \text{!}t\oplus!i > 0 \) then \( \text{count} := \text{!}count + 1; \)
  \( i := \text{!}i + 1 \);
pack \( \text{count} \);
let \( u = \text{new} \lambda o. \langle 0; \text{true}; \emptyset \rangle[\text{!count}] \) in
\( i := 0; \)
while \( \text{!}i \leq n \) do
  (if \( \text{!}t\oplus!i > 0 \) then \( u\oplus!j := \text{!}t\oplus!i; j := \text{!}j + 1 \))
  \( i := \text{!}i + 1 \)
Introduction

Core Language

Pointer Arithmetic

Memory Separation
  With One Heap
  With Multiple Memories
  Linking Memories
  Example

Conclusion
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: $IsHeap$

\[
\forall h. \; IsHeap(h) \implies Global\_ownership\_properties(h)
\]
Another point of view: assumptions

The following code:

\[ x := !y \]

becomes, at the Boogie or Why level:

\begin{verbatim}
assume Global_ownership_properties(h)
h := store(h, x, select(h, y))
assume Global_ownership_properties(h)
\end{verbatim}
Suppose a free variable $x$.

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

Post-condition: $!y >!x$

Resulting proof obligation (simplified):

\[
\forall h, x, y \\
\text{IsHeap}(h) \\
\text{closed}(h, y)
\quad \implies \quad \text{select}(h, y) > \text{select}(h, x)
\]
The heap $h$ is split into several maps from pointers to values.

$$\forall h_x, h_y, x, y \quad \text{IsHeap}(???), \text{closed}(h_y, y) \implies select(h_y, y) > select(h_x, x)$$
Linking Memories (1/2)

Axiom:

\[
\forall h, h_x, h_y .
\begin{align*}
\text{linked}(h, h_x) \\
\text{linked}(h, h_y) \\
\text{IsHeap}(h)
\end{align*}
\] \implies \text{Global_ownership_properties}(h, h_x, h_y)

Proof obligation:

\[
\forall h, h_x, h_y, x, y 
\begin{align*}
\text{linked}(h, h_x) \\
\text{linked}(h, h_y) \\
\text{IsHeap}(h) \\
\text{closed}(h_y, y)
\end{align*}
\] \implies \text{select}(h_y, y) > \text{select}(h_x, x)

This is inconsistent!
Solution: use the assumption point of view.

...some code...

\textbf{assume} \textit{Global\_ownership\_properties}(\textit{Current\_heap}_x, \textit{Current\_heap}_y)

...some code...

Proof obligation:

\begin{align*}
\forall h_x, h_y, x, y \\
\textit{Global\_ownership\_properties}(h_x, h_y) \\
\textit{closed}(h_y, y) \quad \Rightarrow \quad \textit{select}(h_y, y) > \textit{select}(h_x, x)
\end{align*}
Example

... 

pack count;
...

while !i ≤ n do
  (if !t⊕!i > 0 then (u⊕!j :=!t⊕!i; j :=!j + 1))
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
Introduction

Core Language

Pointer Arithmetic

Memory Separation

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, ...