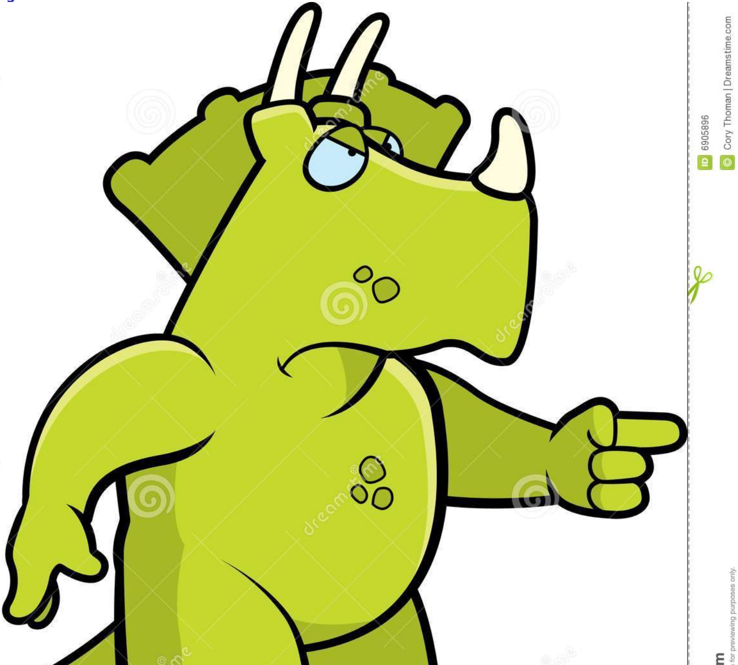
It's Pointless to Point in Bounded Heaps

F.S.de Boer (CWI & LIACS)
Joint work with
Marcello Bonsangue and Jurriaan Rot (LIACS)

Pointer Analysis: Lebende Fossilien?



Since The Dawn of Computer Science

Still Worthwhile To Point Out That:

> *Programs with dynamic linked data structures restricted to bounded heaps are equivalent to programs with only unbounded object creation.*

That is,

> *when restricted to bounded heaps we can remove fields from OO programs.*

That is,

> *bounded heaps is a contradictio in adiecto.*

Outline

1. Symbolic pushdown automata for unbounded object creation.
2. Pushdown automata for bounded visible heaps.
3. Reduction to unbounded object creation

Pushdown Automata

$$(P, \Gamma, \delta)$$

where

- P is a finite set of *control locations*
- Γ is a finite *stack alphabet*
- $\Delta \subseteq (PX\Gamma)X(PX\Gamma^*)$ is a finite set of *transitions*:

$$(p, \gamma) \rightarrow (p', \gamma')$$

Model-checking LTL properties of Pushdown Automata:

A. Bouajjani, J. Esparza, O. Maler.
Reachability Analysis of Pushdown Automata: Application to Model-Checking.
Concur 1997.

Unbounded Object Creation: Even Without Fields It's All Too Much

Even this very simple program

$$m\{u = new; m\}$$

we cannot model-check (unbounded stack alphabet):

Level	*Heap*
n	$u \mapsto n+1$
$\vdots$	
1	$u \mapsto 2$
0	$u \mapsto 1$

A Core Language for Unbounded Object Creation

Statements

S	$::=$	$x := y$	Assignment
	$\mid$	$x :=$ **new**	Object creation
	$\mid$	P	Procedure call
	$\mid$	$S; S$	Sequential composition
	$\mid$	$[x = y]S$	Equality test
	$\mid$	$[x \neq y]S$	Inequality test
	$\mid$	$S + S$	non-deterministic choice

Symbolic States

Conjunction of equations

$$\varphi \ ::= \ x = y \ \mid \ \varphi \wedge \varphi$$

Strongest Postcondition Calculus

Assignment

$$SP(x := y, \varphi) = \varphi[z/x] \wedge x = (y[z/x])$$

Object creation

$$SP(x := \textbf{new}, \varphi) = \varphi[z/x]$$

Procedure call

$$SP(P, \varphi) = \varphi[\bar{z}/\bar{l}] \wedge \bigwedge_{g \in G} (g = g') \wedge \bigwedge_{l \in L \setminus G'} l = \textbf{nil}$$

Return

$$SP(\textbf{ret}\ \psi, \varphi) = \varphi[\bar{z}'/\bar{l}][\bar{z}/\bar{g}'] \wedge \psi[\bar{z}/\bar{g}]$$

Symbolic Transition System

Assignment and Object creation

$$\langle \varphi, B; \mathcal{S} \rangle \longrightarrow \langle SP(B, \varphi) \downarrow_V, \mathcal{S} \rangle$$

where B is either $x :=$ **new** or $x := y$.

Procedure call

$$\langle \varphi, P; \mathcal{S} \rangle \rightarrow \langle SP(P, \varphi) \downarrow_V, B; \varphi; \mathcal{S} \rangle$$

Return

$$\langle \varphi, \psi; \mathcal{S} \rangle \rightarrow \langle SP(\textbf{ret}\ \psi, \varphi) \downarrow_V, \mathcal{S} \rangle$$

An Example Symbolic Computation

Initial configuration $\langle (l = g), P \rangle$, where $P :: l := \textbf{new}$ (l local).

Strongest postcondition procedure call

$$SP(P, l = g) = z = g \wedge g = g' \wedge l = \text{nil}$$

Transition procedure call

$$\langle (l = g), P \rangle \longrightarrow \langle (g = g' \wedge l = \text{nil}), l := \textbf{new}; (l = g) \rangle$$

Strongest postcondition object creation

$$SP(l := \textbf{new}, (g = g' \wedge l = \text{nil})) = g = g' \wedge z = \text{nil}\,,$$

Transition object creation

$$\langle (g = g' \wedge l = \text{nil}), l := \textbf{new}; (l = g) \rangle \longrightarrow \langle (g = g'), (l = g) \rangle\,.$$

Strongest postcondition return

$$SP(\textbf{ret}\ l = g, g = g') = l = z \wedge g = z\,.$$

Transition return

$$\langle (g = g'), (l = g) \rangle \longrightarrow \langle (l = g), \epsilon \rangle$$

Correctness

Details in

Jurriaan Rot, Frank S. de Boer, Marcello M. Bonsangue: Unbounded Allocation in Bounded Heaps. FSEN 2013: 1-16 (extension submitted to special issue of SCP).

Managing Fields

Key idea

Bounded visible heaps allow reuse of strictly local object id's:

$$H = \underbrace{(L \setminus C)}_{\text{reuse}} \cup G$$

where

- H: Heap
- L: Local variables
- C: Cut-points
- G: Global variables

Bounded Heaps: Formal Semantics

Procedure Call

$$\langle h, P; \Gamma\rangle \longrightarrow \langle h[\bar{l} := \bar{0}][C := \textit{cut-points}(h)], B; h; \Gamma\rangle$$

where $\textit{cut-points}(h) = (\mathcal{R}_h(L) \cup \mathcal{R}_h(h(C))) \cap \mathcal{R}_h(G)$.

Return

$$\langle h, h'; \Gamma\rangle \longrightarrow \langle h[\bar{l} := h'(\bar{l})][C := h'(C)], \Gamma\rangle$$

Bounded Heap

$$|\mathcal{R}_h(h(C)) \cup \mathcal{R}_h(L) \cup \mathcal{R}_h(G)| \leq \text{bound}$$

An Example

$$
\begin{array}{ll}
main & :: l := \textbf{new}; g := l; P_1 \\
P_1 & :: P_2; \textbf{while true } \{\textbf{skip}\} \\
P_2 & :: (g.f := \textbf{new}; g := g.f; P_2) + \textbf{skip}
\end{array}
$$

A computation

$$
\begin{array}{ll}
\textit{Level} & \textit{Heap} \\
n & g \mapsto n, n.f \mapsto n+1, C \mapsto \{n+1\} \\
\vdots & \\
1 & g \mapsto 2, 1.f \mapsto 2, C \mapsto \{2\} \\
0 & l, g \mapsto 1, C \mapsto \{1\}
\end{array}
$$

Modeling Bounded Dynamic Linked Data-Structures

- Global canonical representatives: $\bar{1}, \ldots, \bar{n}$
- Field representation: $\bar{i}_f$, for $i = 1, \ldots, n$, field f
- Cut-point representation: local variables c_i, for $i = 1, \ldots, n$

Encoding

- For every global variable g there exists an $\bar{i}$ such that

$$h(g) = h(\bar{i})$$

- For every field f and canonical representative $\bar{i}$

$$h(f)(h(\bar{i})) = h(\bar{i}_f)$$

Reachability

$$
\begin{array}{rl}
R_X = & \Pi_{i=1}^{n} \bar{i}_b := \textbf{false}; \\
& \Pi_{x \in X} \textbf{if } x \neq \textbf{nil then } \Sigma_{i=1}^{k} [\bar{i} = x] \bar{i}_b := \textbf{true}; \\
& (\Pi_{i=1}^{n} S_i)^n
\end{array}
$$

where S_i denotes the statement

$$
\text{if } \bar{i}_b \text{ then } \Pi_{f \in F} \Sigma_{j \in I} [\bar{i}_f = \bar{j}] \bar{j}_b := \textbf{true} \text{ fi}
$$

Translation

Conditional statements

$$t([x = y]S) = [x = y]t(S)$$

Field assignment

$$t(x.f := y) = \Sigma_{i=1}^{n}[\bar{i} = x]\bar{i}_f := y$$

Object creation

$$t(x := \mathbf{new}) = x := \mathbf{new}; R_V; \Sigma_{i=1}^{n}[\neg\bar{i}_b](\bar{i} := x; I_F)$$

where $I_F = \Pi_{f \in F}\bar{i}_f := \mathbf{nil}$.

Procedure Call and Body

Call:

$$t(P) = \quad copy; R'_{L\cup C}; R_G; cutpoint; locals \\ P; \\ R_{C\cup G}; restore$$

where

- $copy :: \Pi_{i=1}^{n}(\bar{i}' := \bar{i}; \Pi_{f\in F}\bar{i}'_f := \bar{i}_f)$
- $cutpoint :: \Pi_{i=1}^{n}\text{if } i'_b \wedge i_b \text{ then } c'_i := \bar{i} \text{ else } c'_i := \textbf{nil} \text{ fi}$
- $locals :: \Pi_{i=1}^{n} i'_b := i'_b \wedge \neg i_b$
- $restore :: \Pi_{i=1}^{n}\text{if } i'_b \text{ then } \Sigma_{j=1}^{n}[\neg j_b]\bar{j} := \bar{i}'; \Pi_{f\in F}\bar{j}_f := \bar{i}'_f$

Body:

$$t(P :: S) = P :: \Pi_{i=0}^{n} c_i := c'_i; t(S)$$

Conclusion

It's pointless to point in bounded heaps.

Literature

- N. Rinetzky, J. Bauer, T. Reps, M. Sagiv, R. Wilhelm. A semantics for procedure local heaps and its abstractions. POPL 2005.
- A. Bouajjani, S. Fratani, S. Qadeer. Context-Bounded Analysis of Multithreaded Programs with Dynamic Linked Structures. Proc. of Computer Aided Verification (CAV 2007).
- Jurriaan Rot, Frank S. de Boer, Marcello M. Bonsangue: Unbounded Allocation in Bounded Heaps. FSEN 2013: 1-16.
- Irina Mariuca Asavoae, Frank S. de Boer, Marcello M. Bonsangue, Dorel Lucanu, Jurriaan Rot: Bounded Model Checking of Recursive Programs with Pointers in K. WADT 2012: 59-76
- Jurriaan Rot, Irina Mariuca Asavoae, Frank S. de Boer, Marcello M. Bonsangue, Dorel Lucanu: Interacting via the Heap in the Presence of Recursion. ICE 2012: 99-113