JAVA and COSY Technology for Embedded Systems (JOSYS)

Supported by the ESPRIT LTR Project #28198

Universität des Saarlandes

Peter Land

Object Inlining
Must be semantically preserving.

We want: automatically detect aggregation possibilities.

But has different semantics:

- Keeps (possibly) related objects together in memory.
- Less objects:
  - Less pointers dereference:
    - Aggregated representation gives
      - Pointer to the heap, pointing to actual data.
      - Uniform representation of data.

What is object initialisation?
Conclusions.

- Constant objects.
- Details of interpreter-based analysis.
- Further improvements (if I have time).
- Representing the objects in memory.
- Analyses, based on this semantics.
- (Informal) semantics for the heap.

Structure of the talk
The semantics of statements is defined obviously.

Inlined objects may not be referenced from reference fields.

contains another object \( \Leftarrow \)

- inlined object

contains a reference to an object \( \Leftarrow \)

- a reference \((I_1, f_2, f_3)\)

- an atomic value \((a_1, a_2, a_3)\)

• A field may be

• An object is just a set of fields

contains references to objects \( \Leftarrow \)

Local and global variables

Definition: semantics of the memory
A transformation is semantics-preserving if it does not change $\text{Ker } \mathcal{E}$.

The heap defines a function from pointer chains to objects: $\mathcal{E}$

- A pointer chain is either a pointer chain followed by a reference or inline field.
- A pointer chain is either a reference variable or

Definition: semantics-preserving
Assignment can be either reference assignment or deep copy:

\( \ell = f \)  

- For field stores (assignments) \( x \):

\[ \text{is the created object going to be initialized?} \]

- For the creation points of objects (statement points) \( x = \text{new } C(\ldots) \):

\[ \text{result — a set of pairs } \langle \text{creation point, reference field} \rangle \]

\[ \text{Could it be an initialized field instead?} \]

\[ \text{For the reference fields:} \]

The analysis should tell us:
An object cannot exist in several copies (1/2)
Visibility of \( \top \) • 

May-alias vs. must-alias (of \( \top \) and \( \bot \))

Differences from previous slide:

- Use \( \top \) use
- \( \exists x. \top = \top \)
- \( \top = \exists x. \top \)
- An object cannot exist in several copies (2/2)
An inlined field cannot contain several objects

\[ \ell \]

\[ x = \text{new } C() \]

\[ z = x1.f \]

\[ \ell.f \text{ is alive} \]

\[ \text{Creations}(x) = \{ \ldots, \ell, \ldots \} \]

\[ \text{Creations}(x1) = \{ \ldots, \ell, \ldots \} \]

\[ z \text{ loaded from } \{ \ldots, \ell, \ldots \}.f \]

use \( z \)

no inlining
Thus it is not allowed to inline an object into its own field.

\[
\begin{align*}
\text{fields}(o) \\
= & \langle \text{size} \rangle + (\text{size}) \text{overhead} \\
\text{size} \quad \sum \\
\text{creation}(f) \quad \text{size} \quad \text{max} \\
\text{is initialized} \quad \text{is initialized} \\
\end{align*}
\]

Where are those objects created, that it points to?

- Type inference tells for each field:
  - Each field has a size.
  - Object is a sequence of fields.
  - Objects are not just sets of fields.
compatible may mean

\( \langle x, \mathcal{I}, \mathcal{F} \rangle \) : \( \mathcal{L} \) new D1()
\( \cdots \)

\( \langle x, \mathcal{I}, \mathcal{F} \rangle \) : \( \mathcal{L} \) new D2()
\( \cdots \)

\( \langle x, \mathcal{I}, \mathcal{F} \rangle \) : \( \mathcal{L} \) new C()
\( \mathcal{L} \)
Incompatibilities are resolved by outlining some fields or by cloning.

The results of this DFA are examined for incompatibilities.

A data flow analysis records

OOPSTA’98, P1D1.00.

We are going to use existing techniques (J. Dolby & A. Chin, P1D1.97,' May be decided at runtime, but this is inefficient.

Is a field store \( x \cdot f = y \) a reference assignment or a deep copy?

At which offset is the field \( f \) in expression \( x \cdot f \)?

Accessing objects in memory
\{c < \text{int} \neq c, c\} = (x) \in \Psi I_L

\return (\text{a})

\{c\} = (x) \in \Psi I_L

\text{new} C() ;

\\text{class} C

\{\ldots \text{d} \neq \ldots\} \text{class} C

\{c < \text{int} \neq c, c\} = (a) \in \Psi I_L
Lattice homomorphism. \( \mathcal{J} \leftarrow \mathcal{J} : A \) \( \to \) is an upper semi-
\( \mathcal{J} \leftarrow \mathcal{J} \) - The result of effect calculation \( \mathcal{J} \) - has a (quite) small set of generators.
For bit-vector analyses: •

\( \mathcal{J} \leftarrow \mathcal{J} \) \( \leftarrow \) \( \mathcal{J} \) \( \leftarrow \) \( \mathcal{J} \) \( \leftarrow \) corresponds to replacing with \( \mathcal{J} \)
Effect calculation analyses the procedure once for each call context.

•

Each program point.
Let the analyses assign an element of the (upper) semilattice to \( \mathcal{J} \) •

The same point that it was called from.
Interpro. controls how graphs does not reflect that each method returns

a.k.a. Functional approach a la Shatir & Platill.

Interprocedural analyses — effect calculation
Clone the method containing call \( \text{good} \) if \( \text{context} \text{call} \text{good} \text{and bad context} \). then replace \( \text{call} \text{good} \) by \( \text{call} \text{good} \): •

Create a copy of \( \text{call} \text{good} \) called \( \text{call} \text{good} \): •

Let a transformation of a method \( \text{method} \) be valid for calling context. Then expression for \( \text{method} \) valid. For cloning \( \text{good} \text{and invalid for} \text{bad} \). For cloning:

Solution — reduce polymorphism.

- Polymorphism harder.
- Code reuse makes program analysis and transfor...
non-constant \[ \Leftrightarrow \]

Object created at \( j \) is modified after being loaded

\[
\forall \ x. \ j = x
\]

\[
...\]

\[
\exists \ z = x
\]

\[
? \) \ new c ( ) : j
\]

Then exist.

As they are only read, it does not matter how many copies of Constant objects may be inlined, even if this is not semantically preserving.

A constant object is not changed after it has been stored.
Conclusions and open problems

Open questions:

- The cost of object initialising analysis is not prohibitive.
- A language with uniform object model is no less efficient than one with explicit aggregation.
- I wanted to convince you that

Multi-threading.

Arrays.

Multi-threading.
<table>
<thead>
<tr>
<th>Implementation status</th>
<th>Program transformation</th>
<th>Representing objects in memory</th>
<th>Interaction of analyses based on set-based semantics</th>
<th>Constant objects</th>
<th>Cloning</th>
<th>Type inference</th>
<th>Bit-vector analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>