Automated Discovery of Scoped Memory Regions for Real-Time Java

Morgan Deters          Ron K. Cytron

Distributed Object Computing Laboratory
Washington University
St. Louis, MO

International Symposium on Memory Management
20 June 2002
Java vs. Real-Time

- High-Level
- Popular
- Object-Oriented
- Garbage Collection

- Reusable Components
- Scalable

- Low-Level
- Specialized
- Manually Optimized
- Specialized Allocators

- Tailored Components
- Doesn’t Scale
Real-Time Java?

- JSR001 *The Real-Time Specification for Java*
- Real-Time Threads
- Asynchronous Transfer of Control
- Scoped Memory Regions

```
MemoryArea
  
  ImmortalMemory

  ScopedMemory

  HeapMemory

  LTMemory

  VTMemory
```
Inter-Object Reference Rules

An object shall not refer to an object from a shorter-lived memory region!

- Objects may only refer to objects in
  - ancestor memory regions
  - garbage-collected heap
Inter-Object Reference Rules

An object shall not refer to an object from a shorter-lived memory region!

- Objects may only refer to objects in
  - ancestor memory regions
  - garbage-collected heap
Using Real-Time Java Scoped Memory

- Consider class `C`:

```java
class C {
    void foo() {
        bar();
        // ...
    }
    void bar() {
        // generate lots of garbage
    }
}
```
Using Real-Time Java Scoped Memory

- `foo()` calls `bar()` in a separate region

```java
class C {
    void foo() {
        LTMemory lt = new LTMemory(1024,1024);
        lt.enter(new Runnable() {
            public void run() {
                bar();
            }
        });
    }
    void bar() {
        // generate lots of garbage
    }
}
```
Using Real-Time Java Scoped Memory

class C {
    int barResult;    // assume single-threaded
    void foo() {
        LTMemory lt = new LTMemory(1024, 1024);
        lt.enter(new Runnable() {
            public void run() {
                barResult = bar();
            }
        });
    }
    int bar() {
        // generate lots of garbage
        // return an int
    }
}
Using Real-Time Java Scoped Memory

class C {
    Object barResult; // assume single-threaded
    void foo() {
        LTMemory lt = new LTMemory(1024, 1024);
        lt.enter(new Runnable() {
            public void run() {
                barResult = bar();
            }
        });
    }
    Object bar() {
        // generate lots of garbage
        // ... new MyObject() ...
    }
}

Illegal reference
Allocated in scoped memory lt

Washington University
St. Louis

20 June 2002
Using Real-Time Java Scoped Memory

```java
class C {
    Object barResult; // assume single-threaded
    void foo() {
        LTMemory lt = new LTMemory(1024,1024);
        lt.enter(new Runnable () {
            public void run() {
                barResult = bar();
            }
        });
        Object bar() {
            // generate lots of garbage
            // ... new MyObject() ...
        }
    }
}
```
RT-Java Memory Summary

• Assign memory regions to execution scopes
• `new` operations allocate from current region

• Introduces memory management dependencies between caller and target
• Hurts reusability of code
• *Integrates* concerns
Goal:

\[ O \rightarrow F \]

\( O \): objects of program \( P \)

\( F \): call-stack frames of program \( P \)
Determine Referencing and Object Lifetimes

- Keep a \textit{doesReference} graph

\begin{tikzpicture}
  \node[draw,shape=circle] (A) at (0,0) {A};
  \node[draw,shape=circle] (B) at (1,0) {B};
  \node[draw,shape=circle] (C) at (2,0) {C};
  \node[draw,shape=circle] (D) at (0,-1) {D};
  \node[draw,shape=circle] (E) at (1,-1) {E};
  \node[draw,shape=circle] (F) at (2,-1) {F};
  \node[draw,shape=circle] (G) at (1,-2) {G};

  \draw[->] (A) to (B);
  \draw[->] (B) to (C);
  \draw[<-] (D) to (E);
  \draw[<-] (E) to (F);
  \draw[<-] (E) to (G);
\end{tikzpicture}

- Annotate it with object lifetimes (\# of frames)
Determine Referencing and Object Lifetimes

• Keep a *doesReference* graph

• Annotate it with object lifetimes (# of frames)
Solution to *doesReference* graph

- Decompose into strongly-connected components
Solution to *doesReference* graph

- Decompose into strongly-connected components
Solution to *doesReference* graph

- Propagate smaller numbers through the graph
Solution to *doesReference* graph

- Propagate smaller numbers through the graph

![Diagram with nodes A, B, C, D, E, F, G and edges with numbers 0 and 2]
Q: How to recognize objects from one run to another?

A: Use an object’s *allocation site* as a discriminator.
Impact of Simplification

• Consider a method that acts as a Factory:

```java
class LinkedList {
    public Iterator iterator() {
        return new LLIterator();
    }
}

Iterator i = list.iterator();
while(i.hasNext()) {
    // ...
}
```
Impact of Simplification

- Consider a method that acts as a Factory:

```java
class LinkedList {
    public Iterator iterator() {
        return new LLIterator();
    }
}

Iterator i = list.iterator();
while(i.hasNext()) {
    // ...
}
```
Revised Goal:

\[ S \rightarrow \Delta F \]

\( S: \) allocation sites of program \( P \)

\( F: \) frames of program \( P \)

\( \Delta F: \) object lifetimes in \# \ of frames
Determine Referencing and Object Lifetimes
Solving the Graph

• Liveness constraints:
  For each vertex $v$
  \[ \text{Scope}(v) = \max_{o \in \text{Objects}(v)} o_i - o_c \]

  $o_i = \text{object allocation frame}$
  $o_c = \text{object collection frame}$

• Reference constraints:
  For each edge $(u,v)$
  \[ \text{Scope}(v) + \text{Scope}(u) + \max_{o \in \text{Objects}(v)} o_i - \min_{o \in \text{Objects}(u)} o_i \]
Solution to doesReference graph
Experimentation

• Instrumented JVM 1.1.8 to emit
  ▪ object allocations
  ▪ object collections
  ▪ intra-heap references

• SPECjvm98 benchmarks
Compress assigned scopes

Scopes Assigned to Objects at Runtime

# Objects

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Compress extra longevity

![Graph showing compress extra longevity]
Raytrace assigned scopes

Scopes Assigned to Objects at Runtime

# Objects
Raytrace extra longevity

Extra longevity - _205_raytrace

# Frames

# Objects
Db scopes assigned
Db extra longevity
Jack scopes assigned
Jack extra longevity

Extra Longevity - __228_jack

# Frames

# Objects

# Frames
Automated Discovery of Scoped Memory Regions for Real-Time Java

_Morgan Deters_

Washington University
St. Louis

20 June 2002
Ongoing Work

• Shared scopes
• Tool support
• Further evaluation…timing
• And…
Combining Static and Dynamic Analyses

• Static analysis used for
  ▪ Eliminating RTSJ-mandated reference checks
  ▪ Verifying scope assignments, flagging others for programmer input

• Dynamic analysis used for
  ▪ Initial scope assignment
  ▪ Debugging information
Related Work

• Regions
  ▪ Tofte & Talpin
  ▪ Gay & Aiken

• Real-Time Java analysis
  ▪ Pointer and escape analysis for Java
  ▪ Sălcianu & Rinard
Remarks

• Garbage collection is great, but…
• Stack-based region-allocated objects
• RTSJ Scope model breaks modularity
• Compose with results of static analysis
• Extension to non-RTSJ systems