Verifying Memory Properties of SCJ Programs

Chris Marriott

The University of York

29th November 2011
Safety-Critical Java (SCJ)

Memory safety in SCJ

Checking techniques

Existing approaches for SCJ

SCJ-Circus

Conclusion and future work
Safety-Critical Java

- A new programming paradigm
  - Mission and handler concept
  - Not necessarily restricted to Java

- Designed specifically for “safety-critical” applications
  - Level 0 - Cyclic executive
  - Level 1 - Concurrent handlers (comparable to Ravenscar Ada)
  - Level 2 - Concurrent missions and handlers

- All levels certifiable to DO-178B Level A (depending on funds!)
Programming paradigm

- Safelet
- Mission Sequencer
- Missions
- Handlers
Java memory management

**Java**
- Heap
- Garbage collection
- No concern for the programmer

**RTSJ**
- Heap and Scoped memory areas
- No garbage collection on scoped areas

**SCJ**
- No heap
- Scoped memory areas
- No garbage collection
Scoped memory model

Immortal memory
- Lasts for the entire length of the program
- Objects cannot be removed

Scoped memory
- Have a specific lifetime (Mission, Handler, Temporary)
- Areas are cleared out at the end of the execution
1 Safety-Critical Java (SCJ)

2 Memory safety in SCJ

3 Checking techniques

4 Existing approaches for SCJ

5 SCJ-Circus

6 Conclusion and future work
Memory safety in SCJ

Thread Stacks (one per ASEH and one each for the mission sequencer and main program)

Key:
- Valid object references
- an illegal reference
Scoped memory introduces the possibility of dangling references

Allocation must be thought about

References must not point **down** the memory structure
1. Safety-Critical Java (SCJ)

2. Memory safety in SCJ

3. Checking techniques

4. Existing approaches for SCJ

5. SCJ-Circus

6. Conclusion and future work
Correctness by construction

- Abstract specification
- Refinement laws
- Resulting program is correct because of the soundness of the refinement technique used

Program / Property verification

- Assertions
- Code analysis
- Static / dynamic techniques
Checking techniques

Static checking
- Model checking
- Data-flow analysis
- Thorem proving

Dynamic checking
- Execution-based analysis
- Code coverage problem
1. Safety-Critical Java (SCJ)

2. Memory safety in SCJ

3. Checking techniques

4. Existing approaches for SCJ

5. SCJ-Circus

6. Conclusion and future work
Existing approaches for SCJ

- SafeJML
- $R_{sj}$
- SCJ Checker
SafeJML

- Extension to the classic JML annotation language
- Focused heavily on timing extensions
- Used to perform WCET analysis
- Scheduling algorithm for use with Java Pathfinder
- JPF is a model checker
- Fixed-priority pre-emptive scheduler
- Gives time predictable results based on the JOP
- Cannot handle SCJ Level 1 (AperiodicEvents)
Annotation-based approach

Level, behaviour, and memory annotations

Annotations impose more restrictions than the programming paradigm

- Class instances are locked to defined scopes
- Annotated and unannotated class instances cannot interact
1. Safety-Critical Java (SCJ)

2. Memory safety in SCJ

3. Checking techniques

4. Existing approaches for SCJ

5. SCJ-Circus

6. Conclusion and future work
What can we do differently?

- SCJ Checker doesn’t have a formal base
  - We do not doubt the effectiveness of the technique
  - There is nothing to say it’s sound
- Model checkers can’t handle the state explosion found at Level 1
- Are the restrictions imposed by the annotations necessary?
A **sound** static-analysis technique to verify memory safety

Memory safety $\iff$ No dangling references
SCJ Program

- SCJ code that conforms to the SCJ specification
- Up to Level 1
- Well typed
- Well formed
  - What this actually means will become clear!
- SCJ abstraction
  - Behavioural and functional properties maintained
  - Concurrency maintained
  - No timing properties
  - No scheduling properties
  - No storage requirements
- Consistent structure for SCJ paradigm
- Unique naming throughout
Example

SCJ

class safelet implements Safelet {
    static Object o = new Object();
    int a;

    public void setUp() {
        int b;
    }

    public MissionSequencer getSequencer() {
        priority = new PriorityParameter(PriorityScheduler.instance().getNormPriority()),
        storage = new StorageParameters(100000L, 1000, 1000));
        return new missionSequencer(priority, storage);
    }

    public void tearDown() {
        int c;
    }
}

SCJ-M

static {Object o}
sInit { o = new Object(); }
safelet {
    fields { int a; }
    constr { skip; }
    setUp { int b; }
    tearDown { int c; }
}
Example

SCJ

class handler1 extends PeriodicEventHandler {
    int v1;
    int v2;
    
    public handler1(PriorityParameters p, PeriodicParameters r,
    StorageParameters s) {
        super(p, r, s);
        v1 = 50;
        v2 = 100;
    }

    public void handleAsyncEvent() {
        if (v1 < v2) { v1++; }
        else { v2++; }
    }
}

SCJ-M

handler handler1 {
    fields {
        int v1;
        int v2;
    }
    constr {
        v1 = 50;
        v2 = 100;
    }
    hAe {
        if (v1 < v2) { v1++; }
        else { v2++; }
    }
}
SCJ-M gives a syntactic description of SCJ programs

Inference rules check the syntactic description

SCJ-M semantics required to establish soundness of technique
**Verifying Memory Safety**

\[ m\text{safe}_{\text{syn}} \]

\[ m\text{safe}_{\text{syn}} = SCJ\_M \rightarrow \text{Boolean} \]

\[ m\text{safe}_{\text{utp}} \]

\[ m\text{safe}_{\text{utp}} = UTP\text{Relations} \rightarrow \text{Boolean} \]

\[ m\text{safe} \]

\[ m\text{safe}_{\text{syn}}(\text{program}) \Leftrightarrow m\text{safe}_{\text{utp}}(\llbracket \text{program} \rrbracket) \]
Environments are required to store the current variables in scope

Mapping from variable names to creation contexts

\[
\text{CreationCon} ::= \text{Prim} | \\
\text{IMem} | \\
\text{MMem} | \\
\text{PRMem} \langle \langle \text{Hid} \rangle \rangle | \\
\text{TPMem} \langle \langle \text{Hid} \times \mathbb{N} \rangle \rangle | \\
\text{TPMMem} \langle \langle \text{Mid} \times \mathbb{N} \rangle \rangle
\]

\[
\text{VEnv} ::= \text{VName} \rightarrow \text{CreationCon}
\]
\[ m\text{SAFE}_{syn} = SCJ_M \rightarrow Boolean \]

- Inference rules based on the memory safety properties found in the SCJ specification

\[
\frac{m\text{SAFE}_{e1}(P.\text{sl}\text{Init}, \text{iMem}) \quad m\text{SAFE}_{e2}(P.\text{safelet})}{m\text{SAFE}(P)} \quad (1)
\]

where
\[
e1 = \text{CalcE}(\emptyset, P.\text{static}, \text{iMem})
\]
\[
e2 = \text{CalcEC}(e1, P.\text{sl}\text{Init}, \text{iMem})
\]
$msafe_{syn} = SCJ\_M \rightarrow \text{Boolean}$

\[
msafe_{e_1}(o = \text{new Object()}, I\text{Mem}) \quad msafe_{e_2}(P\_\text{safelet}) \\
\quad msafe(P) \quad (1)
\]

where
\[
e_1 = \text{CalcE}(\emptyset, P\_\text{static}, I\text{Mem}) \\
e_2 = \text{CalcEC}(e_1, P\_\text{sInit}, I\text{Mem})
\]
\[ m\text{safe}_{\text{syn}} = \text{SCJ}_M \rightarrow \text{Boolean} \]

\[
\frac{\text{msafe}_{e_1}(o = \text{new Object()}, \text{IMem}) \quad \text{msafe}_{e_2}(P.\text{safelet})}{\text{msafe}(P)}
\] (1)

where

\[ e_1 = \{o \mapsto \text{IMem}\} \]
\[ e_2 = \text{CalcEC}(e_1, P.\text{sInit}, \text{IMem}) \]
$$m_{safe_{syn}} = SCJ\_M \rightarrow \text{Boolean}$$

$$m_{safe_{e_1}}(o = \text{new Object()}, \text{IMem}) \quad m_{safe_{e_2}}(P.\text{safelet})$$

$$\quad \quad \quad \quad \quad \downarrow$$

$$m_{safe}(P)$$

(1)

where

$$e_1 = \{o \mapsto \text{IMem}\}$$

$$e_2 = \{o \mapsto \text{IMem}\}$$
More rules - Safelet

\[
\begin{align*}
\text{msafe } e_2(S.\text{constr}, \text{IMem}) & \quad \text{msafe } e_3(S.\text{setUp}, \text{IMem}) \\
\text{msafe } e_4(S.\text{missionSeq}) & \quad \text{msafe } e_5(S.\text{tearDown}, \text{IMem}) \\
\hline
\text{msafe } e_1(S)
\end{align*}
\]

where

\[
\begin{align*}
e_2 & = \text{CalcE}(e_1, S.\text{fields}, \text{IMem}) \\
e_3 & = \text{CalcEC}(e_2, S.\text{constr}, \text{IMem}) \\
e_4 & = \text{CalcEC}(e_3, S.\text{setUp}, \text{IMem}) \\
e_5 & = \text{CalcEMS}(e_4, S.\text{missionSeq})
\end{align*}
\]
\[
\begin{align*}
msafe_{e_2}\text{(skip, IMem)} & \quad msafe_{e_3}\text{(int b;, IMem)} \\
msafe_{e_4}\text{(S.missionSeq)} & \quad msafe_{e_5}\text{(int c;, IMem)} \\
\hline
msafe_{e_1}\text{(S)}
\end{align*}
\]

where
\[
\begin{align*}
e_1 & = \{ o \mapsto \text{IMem} \} \\
e_2 & = \text{CalcE}(e_1, S.fields, \text{IMem}) \\
e_3 & = \text{CalcEC}(e_2, S.constr, \text{IMem}) \\
e_4 & = \text{CalcEC}(e_3, S.setUp, \text{IMem}) \\
e_5 & = \text{CalcEMS}(e_4, S.missionSeq)
\end{align*}
\]
More rules - Safelet

\[ m\text{safe}_{e_2}(\text{skip, IMem}) \quad m\text{safe}_{e_3}(\text{int b, IMem}) \]
\[ m\text{safe}_{e_4}(S\.missionSeq) \quad m\text{safe}_{e_5}(\text{int c, IMem}) \]
\[ \frac{m\text{safe}_{e_1}(S)}{2} \]

where
\[ e_1 = \{ o \mapsto \text{IMem} \} \]
\[ e_2 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \} \]
\[ e_3 = \text{CalcEC}(e_2, S\.constr, \text{IMem}) \]
\[ e_4 = \text{CalcEC}(e_3, S\.setUp, \text{IMem}) \]
\[ e_5 = \text{CalcEMS}(e_4, S\.missionSeq) \]
More rules - Safelet

\[
\begin{align*}
\text{msafe}_{e_2}(\text{skip, IMem}) & \quad \text{msafe}_{e_3}(\text{int } b;, \text{IMem}) \\
\text{msafe}_{e_4}(S.\text{missionSeq}) & \quad \text{msafe}_{e_5}(\text{int } c;, \text{IMem}) \\
\hline
\text{msafe}_{e_1}(S)
\end{align*}
\]

where

\begin{align*}
\text{e}_1 & = \{ o \mapsto \text{IMem} \} \\
\text{e}_2 & = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \} \\
\text{e}_3 & = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \} \\
\text{e}_4 & = \text{CalcEC}(\text{e}_3, S.\text{setUp}, \text{IMem}) \\
\text{e}_5 & = \text{CalcEMS}(\text{e}_4, S.\text{missionSeq})
\end{align*}
\[ \text{msafe}_{e_2}(\text{skip, IMem}) \quad \text{msafe}_{e_3}(\text{int } b; , \text{IMem}) \]

\[ \text{msafe}_{e_4}(S.missionSeq) \quad \text{msafe}_{e_5}(\text{int } c; , \text{IMem}) \]

\[ \text{msafe}_{e_1}(S) \] (2)

where

\[ e_1 = \{ o \mapsto \text{IMem} \} \]

\[ e_2 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \} \]

\[ e_3 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \} \]

\[ e_4 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem}, b \mapsto \text{IMem} \} \]

\[ e_5 = \text{CalcEMS}(e_4, S.missionSeq) \]
More rules - Safelet

\[
\begin{align*}
msafe_{e_2}(\text{skip, IMem}) & \quad msafe_{e_3}(\text{int } b;, \text{IMem}) \\
msafe_{e_4}(S.\text{missionSeq}) & \quad msafe_{e_5}(\text{int } c;, \text{IMem}) \\
\hline
msafe_{e_1}(S)
\end{align*}
\] (2)

where

\[e_1 = \{ o \mapsto \text{IMem} \}\]
\[e_2 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \}\]
\[e_3 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem} \}\]
\[e_4 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem}, b \mapsto \text{IMem} \}\]
\[e_5 = \{ o \mapsto \text{IMem}, a \mapsto \text{IMem}, b \mapsto \text{IMem}, \ldots \}\]
More rules - Mission sequencer

\[
\text{msafe}_{e_1}(MEnv(Mid)) \quad \text{msafe}_{e_2}(MS) \\
\text{msafe}_{e_1}(<Mid> \sim MS)
\]  

(3)

where

\[e_2 = CalcEM(e_1, MEnv(Mid))\]
More rules - Declarations and Commands

\[
\text{True} \quad \frac{}{\text{msafe}_e(\text{var}(d), \text{MA})} \quad (8)
\]

\[
\text{True} \quad \frac{}{\text{msafe}_e(\text{skip}, \text{MA})} \quad (16)
\]
More rules - Declarations and Commands

\[
\text{ExpCc}(\text{exp}, \text{e}) \mapsto e(vn) \in m\text{safeRefs} \\
\text{msafe}_e(\text{ass}(vn, \text{exp}), MA) (11)
\]

\[
\text{ExpCc}(\text{exp}, \text{e}) \mapsto e(vn) \in m\text{safeRefs} \\
\text{msafe}_e(a = b, MA) (11)
\]

\[
\text{ExpCc}(b, \text{e}) \mapsto e(a) \in m\text{safeRefs} \\
\text{msafe}_e(a = b, MA) (11)
\]

\[
\text{IMem} \mapsto \text{IMem} \in m\text{safeRefs} \\
\text{msafe}_e(a = b, MA) (11)
\]
\[
\begin{align*}
\text{True} & \quad \text{True} \\
\frac{\text{msafe}_{e_1}(v_1++, \text{PRMem}(\text{handler1.id}))}{11} & \quad \frac{\text{msafe}_{e_1}(v_2++, \text{PRMem}(\text{handler1.id}))}{11} \\
\frac{\text{msafe}_{e_1}(v_1=50, \text{PRMem}(\text{handler1.id}))}{11} & \quad \frac{\text{msafe}_{e_1}(v_2=100, \text{PRMem}(\text{handler1.id}))}{10} \\
\vdots & \quad \vdots \\
\frac{\text{msafe}_{e_1}(v_1=50; v_2=100, \text{PRMem}(\text{handler1.id}))}{6} & \quad \frac{\text{msafe}_{e_1}(\text{if}(v_1<v_2) \ldots, \text{PRMem}(\text{handler1.id}))}{6} \\
\vdots & \quad \vdots \\
\frac{\text{msafe}_\varnothing(\text{handler1})}{5} & \quad \frac{\text{msafe}_{e_1}(\emptyset)}{5} \\
\frac{\text{msafe}_\varnothing(\text{skip, MMem})}{16} & \quad \frac{\text{msafe}_\varnothing(\text{skip, MMem})}{16} \\
\frac{\text{msafe}_\varnothing(\text{mission1})}{3} & \quad \frac{\text{msafe}_\varnothing(\emptyset)}{3} \\
\frac{\text{msafe}_\varnothing(\text{skip, IMem})}{16} & \quad \frac{\text{msafe}_\varnothing(\text{skip, IMem})}{16} \\
\vdots & \quad \vdots \\
\frac{\text{msafe}_\varnothing(\text{safelet})}{16} & \quad \frac{\text{msafe}_\varnothing(\emptyset)}{16} \\
\frac{\text{msafe}_\varnothing(\text{P})}{1} \\
\text{where} & \\
e_1 = v_1 \mapsto \text{Prim}, v_2 \mapsto \text{Prim} \\
\vdots & \quad \vdots \\
\frac{\text{msafe}_\varnothing(\text{mission1})}{7} & \quad \frac{\text{msafe}_\varnothing(\emptyset)}{7}
\end{align*}
\]
1. Safety-Critical Java (SCJ)

2. Memory safety in SCJ

3. Checking techniques

4. Existing approaches for SCJ

5. SCJ-Circus

6. Conclusion and future work
Conclusion and future work

- Syntactic and semantic approach for soundness
- Not all language constructs catered for yet
  - Multi-dimensional data types
  - Method calls
  - Classes and dynamic binding
  - Level 1
- Semantic work yet to be investigated
- Automation of the technique
  - Translation from SCJ to SCJ-M
  - Application of rules