GC Assertions: Using the Garbage Collector to check heap properties

Shirley Gracelyn

February 16, 2011

Motivation

1. Automatic memory management frees some burden from programmers
2. Fewer memory management errors

Motivation

1. Automatic memory management frees some burden from programmers
2. Fewer memory management errors
3. Disadvantages: Does not provide any control or information over behavior of the memory to the programmers
4. Memory which is reachable but which will never be used again might not be freed

Examples

- Is there only a single instance of this type?
- Will this object be reclaimed at the end of next garbage collection cycle?
- Are there any outstanding references to this object?

Main Idea - GC Assertions

- Efficiency of automatic memory management is retained
- Programmer can query the garbage collector for violations in object or data structure properties
- Ensures less time and space overhead in GC assertion checking
- Piggyback checks with usual GC tracing process
- Limits meta data to be stored in extra bits in object headers

Heap properties

- Object life time
- Allocation volume
- Connectivity
- Ownership

Advantages of GC Assertions

- Precise than static analysis

Advantages of GC Assertions

- Precise than static analysis
- Efficient than run time invariant checking

Advantages of GC Assertions

- Precise than static analysis
- Efficient than run time invariant checking
- Accurate than heuristics

Programmer driven approach

- Allows to express and capture behavior of objects which is already known
- Heuristics suggest only potential leaks, which needs to be verified manually
- Rather, with assertions, programmers can specify exactly when the object should be dead, and violations are detected sooner.

Implementation

- Jikes RVM 3.0.0
- Mark sweep collector

Lifetime assertions: assert-dead(p)

- Triggered if object pointed to by p is still reachable
- Usage: To check if a particular object is reclaimed at a specific point in the program
- Implementation: Set a spare bit in the object's header pointed to by p
- Check during next GC cycle if any object has dead bit set

Lifetime assertions: assert-alldead()

- Used along with start-region() assertion
- Triggered if any object allocated in the region, is not reclaimed when the assertion is checked
- Usage: To check if a code section does not leak memory into the rest of the application

Lifetime assertions: assert-alldead()

- Implementation: Thread specific bit - whether the current region is part of an alldead region
- Queue - to store all objects allocated in the alldead region
- On an assert-alldead assertion, the bit is reset.
- assert-dead() is called on each object in the queue

Volume assertions: assert-instances(T,I)

- Triggered if the number of objects of type T goes over I
- Usage: Checking for singleton pattern
- Limit the number of objects for performance reasons

Volume assertions: assert-instances(T,I)

- Implementation: Assertion is tied to object type, and not instance
- Maintain the instance limit and count for the class
- Increment instance count for each object when encountered during GC cycle
- Finally, iterate through list of tracked types to check for instance limit violation

Ownership assertions: assert-unshared(p)

- Triggered if the object pointed to by p has more than one incoming pointer
- Usage: Check if a tree data structure has not accidentally been changed to a DAG or graph
- Implementation: Just set a bit on the object indicating that it should be unshared, which is checked for during garbage collection

Ownership assertions: assert-ownedby(p,q)

- Triggered if object pointed to by q is not owned by object pointed to by p
- Definition of owner and ownee?
- All paths from roots through heap must pass through owner - Too restrictive
- The set of paths through heap to the ownee must include atleast one path that passes through the owner

Ownership assertions: assert-ownedby(p,q)

- Programmer has to identify the larger data structure that governs its life time
- Usage: Assertion ensures that the object will never outlive its owner
- Restriction to lower cost over head - Regions of the heap belonging to different owners should not overlap

Ownership assertions: assert-ownedby(p,q)

- Implementation: On encountering the ownee, GC can check if the owner was present in this path
- But,what if the owner is followed by a previously marked object?
- Repeating the tracing information to check if ownee is reachable from previously marked object is not a good idea
- Bubbling up the ownee information throughout causes unnecessary space and time overhead

Ownership assertions: assert-ownedby(p,q)

- Ownership phase: Start GC tracing with owner objects, assuming the owners themselves are alive
- On visiting the ownee object verify if it belongs to the current owner, else issue a warning
- If another owner is encountered, stop the scan. This owner will be scanned separately
- Normal heap scan: Start from the roots, any ownee encountered is not properly owned

On violation of assertions

- Log an error, but continue executing
- Log an error and halt
- Force the assertion to be true

Debugging information

- Report provides full path through object graph from root to dead object
- Limitation: Identifying the offending object or path is not possible in all cases

Evaluation

- Example 1: Order processing system - destroy() is called on Order object
- assert-dead assertion on Order objects failed
- Reason? Customer objects still had a last order field holding the reference to Order object
- Solution: Set back reference pointer to null

```
Path: Lspec/jbb/Customer; ->
[Ljava/lang/Object; ->
Lspec/jbb/LastOrder; ->
[Ljava/lang/Object; ->
Lspec/jbb/Order;
```

Evaluation

- Example 2: Local variable retains a reference to Company object from the previous iteration
- Sample of a memory drag, as object will be reclaimed in next iteration
- Can be detected through calls to assert-instances as more than one Company instance should not be alive at the same time

Limitations

- To insert assert-dead assertion at the right place, programmer should know when the object should be dead, i.e where the objects become unreachable
- Easier option : ownership assertion
- Fixed assertion set - limited expressiveness
- Checks are not done immediately - incorrect evaluation due to changes in heap