A Concurrent Incremental Run–Time Invariant Checker for Java

Chenyu Zheng
CSCI 5535 – Spring 2009
Fundamentals of Programming Languages
Content

- Goal
- Assertions in Java
- Thoughts from Garbage Collection
- Checker Design
Goal

- Invariant checks were expensive because of the performance penalty caused by stopping the "main thread" of execution

- But, what if invariant checks are running on a separate thread
  - Reduce the disruptiveness of invariant check
  - Provide real-time guarantees
  - Be generalized into concurrent collections

- Design an algorithm that performs concurrent and incremental run-time checking of invariants concurrently
Assertions in Java

- **Forms**

  ```java
  assert Exp1; (Exp1 is a boolean expression)
  assert Exp1 : Exp2; (value of Exp2 is passed to AssertionError constructor)
  ```

- **Rules of Avoiding Side Effects**

  Evaluating the expression should not affect any state that is visible after the evaluation is complete

e.g.

  ```java
  // Broken! - action is contained in assertion
  assert names.remove(null);
  
  // Fixed - action precedes assertion
  boolean nullsRemoved = names.remove(null);
  assert nullsRemoved; // Runs whether or not asserts are enabled
  ```
Assertions in Java Cont.

- Situations of Using Assertions
  - Internal Invariants (including class invariants)
    ```java
    if (i % 3 == 0) {
    ...
    } else if (i % 3 == 1) {
    ...
    } else {
        assert i % 3 == 2 : i;
    ...
    }
    ```
  - Control–Flow Invariants
    ```java
    void foo() {
        for (...) {
            if (...) {
                return;
            }
            assert false; // Execution should never reach this point!
        }
    }
    ```
  - Preconditions and Postconditions
    ```java
    private void setRefreshInterval(int interval) {
        // Confirm adherence to precondition in nonpublic method
        assert interval > 0 && interval <= 1000/\text{MAX\_REFRESH\_RATE} : interval;
    }
    ```
Thoughts from Garbage Collection

- Garbage Detection
- Tricolor Marking
- Read and Write Barriers
The basic functioning of GC consists of

- Garbage Detection
distinguish the live objects from the garbage
- Garbage Reclamation
reclaim the garbage objects’ storage

And we will use garbage detection techniques to find the live set (objects whose states will be checked) of invariant checker
Garbage Detection

- **Root Set**
  active variables at the moment GC is invoked

- **Live Set**
  objects on any directed path of pointers from the roots

- **Done in Two ways**
  - Reference Counting
  - Tracing
Reference Counting

- Maintain counts of the number of pointers to each object
- If reference count of a object becomes zero, this object is reclaimed

Advantage
incremental nature, interleaved closely with the running program’s own execution

Disadvantage
- Not effective in reclaim circular structures
- Difficult to make efficient as proportional to running program’s work
- Counts’ overflow and space wasting
Tracing

- **Procedure**
  - Traversing the pointers from root set
  - Building the live set
  - Mark the live objects
  - Sweep / Compact / Copying / Non-Copying

- **Advantage**
  Accurate and efficient (proportional to the amount of live data for copying GC)

- **Disadvantage**
  - Fragmentation (Sweep)
  - Cost of memory adjustment (Compact, Copying)
  - Require the actual language-level pointers between objects to be changed (Non-Copying)
Incremental Tracing Collectors

- More efficient and effective than incremental reference counting collectors

- Problem
  while the collector is tracing out the graph of reachable objects, the running program (mutator) may mutate the graph

- Solution
  - Tricolor Marking
  - Read and Write Barriers
Tricolor Marking

- Traverse the pointers from the root set
- Black: live objects to be retained
- White: garbage objects
- Gray: objects have been reached by the traversal, but its descendants may not have been
Coordinate Collector with Mutator

- **Read Barrier**
  Detects when the mutator attempts to access a pointer to a white object and immediately colors the objects gray

- **Write Barrier**
  Detects when the program attempts to write a pointer into an object, the write is trapped or recorded
Coordinate Collector with Mutator Cont. (Write Barriers)

- Snapshot-at-beginning Collector
  Rather than allowing pointers to be simply overwritten, they are first saved so that the collector can find them

- Incremental Update Collector (more direct)
  Rather than noticing if the original is destroyed, they actually record pointers stored into black objects and catch the troublesome copies where they are stored
Checker Design (Three-Phase Abstraction)

- Roots Detection
  Determine which variables are the roots of the invariant checker

- Live Sets Marking and Locking
  Mark the live objects by traversing through pointers with tricolor and lock all the live objects

- Concurrent Execution
  Checker and mutator (main thread) can now run in parallel because of the lock scheme
Roots Detection

Main thread of execution is stopped and a new thread is created for the invariant checker, in which:

- If Exp1 only has expression
  only the expressions in Exp1 are roots

- Else Exp1 has function calls
  as in GC, active variables at the moment checker is invoked are all roots

Roots will be used to produce live set (objects whose states will be checked)
Live Sets Marking

- Starting iterative-deepening depth-first search (guarantee the shortest path and save space) from roots by following pointers

- Using tricolor marking scheme, as defined in incremental tracing garbage collector, to mark live objects

Problem
- Mutator reads or writes live objects which are not covered by the marking procedure at the moment
- Mutator destroys the original pointers to live objects before the checker see them
Live Sets Marking Cont.

Solutions

- Read and Write Barrier
  Detects when the mutator attempts to access a pointer to a white object or to write a pointer to a white object and the process of live set marking is not finished, immediately records the action

- Pointer Barrier
  Detects when the mutator are trying to destroy the original pointers to live objects before the checker sees it, records the action

Both Barriers are implemented with incremental update scheme as in incremental tracing garbage collector
Live Sets Locking and Concurrent Execution

- After marking all the live objects, the checker places a lock on all the objects in the live set.

- After locking all the live objects, the main thread of execution continues to run concurrently with the checker thread.
Future work

- Implement concurrent incremental run-time invariant checker in JikesRVM
- Compare performance of the checker with java’s own assertion checker
Questions

- Thank you 😊