Ghosts in the machine

Scheme Workshop 2016

Daniel Szmulewicz

1 Abstract

In Clojure, Vars are a foundational abstraction for naming things and looking up values. They are stable, logical entities. In this talk, I will show how this abstraction leaks under specific circumstances.

In traditional Lisp systems, restarting the REPL is a matter of respawning a process. Not so with Clojure. As a hosted language, it needs to bootstrap itself in the runtime before everything else. REPL restarts are thus costly. As a result, practices have emerged to keep REPL sessions going for as long as possible.

A solution that facilitates bottom-up programming comes in the shape of a Clojure contrib[1] library called `tools.namespace`. It leverages Clojure's built-in "lib" facility, and works by loading modified namespaces in dependency order, after unloading them in reverse order. Usage of this library will yield a seamless live coding workflow, but will induce an unfortunate side-effect in the JVM.

Indeed, when unmapped, Var objects linger in the heap and will eschew garbage collection. This goes largely unnoticed because the redefined Var resolves to the new value, as the user expects. However, a heap dump analysis of a reloaded application will reveal a memory leak (increase in allocated space per Var) of order O(n).

Further complications arise when threads are involved. If a thread references a Var that gets unmapped, that thread will never see the Var's new value. The promise of Vars as stable references withers away. This effect can be countered by adhering to

[1] Clojure Contrib is a collection of libraries managed with the same Contributor Agreement, license, and development workflow as Clojure itself. Code in Clojure Contrib can be considered for inclusion in newer versions of Clojure.

strict coding discipline: restarting all threads when modified code is known to produce orphaned Vars.

The discussion will revolve around the particulars of leaky Vars, workarounds and suggested improvements, but we'll also delve into the inescapable duality between a runtime (a machine) and language semantics.