Weak References

Data Structures and Implementation

Bruno Haible
ILOG GmbH

24 April 2005

What is a Weak Pointer?

- Garbage collection preserves all objects that are *reachable* from the *root set*.
- A *weak pointer* holds its object without causing it to be reachable.

What is a Weak Hashtable?

- A weak hash-table holds its key-value pairs without causing them to be reachable.

- Four kinds:
  - `:key`
  - `:value`
  - `:key-and-value`
  - `:key-or-value`

A Strong Feature

- Adding extra info to sealed objects.
- Memoizing prior results.
- Uniquification.
- Hash consing.
- Avoiding attach/detach protocols.
- Global garbage collection.

Caveats

- Extra time spent in GC
  (for W weak pointers:
  - $O(W^2)$ in some implementations,
  - $O(W)$ in other implementations)

Weak Datastructures

- Weak pointer
- Weak “and” relation
- Weak “or” relation
- Weak association (= weak mapping)
- Weak “and” mapping
- Weak “or” mapping
- Weak association list
- Weak hash-table

Primitive Weak Datastructures

- Weak pointers
- Weak :key mappings
- Weak hash-tables

The others can be emulated.

Levels of Support

1. Support for weak pointers.
2. Support for weak :key mappings or weak hash-tables, with "key not in value" restriction.
3. Support for weak :key mappings or weak hash-tables, without restriction.
4. Scalable support for weak :key mappings or weak hash-tables.

Implementations of Level 1

- Common Lisp: GNU clisp 2.33.80, OpenMCL 0.14.1, Allegro CL 6.2, LispWorks 4.3, Corman Lisp 1.1, ~~CMUCL 19a~~, ~~SBCL 0.8.20~~
- Scheme: GNU guile 1.7.1, MIT Scheme 7.7.1, BBN Scheme, MzScheme 205, Scheme48
- Other Lisp: XEmacs 21.4, GNU Emacs 21.1, jlisp 1.03, mindy 1.2
- Java 1.5
- .NET CLR (mono 1.0.1, ~~pnet 0.6.10~~)
- Smalltalk: GNU Smalltalk 2.1.10
- ~~Python 2.4~~

Implementations of Level 2

- Common Lisp: GNU clisp 2.33.80, OpenMCL 0.14.1, Allegro CL 6.2, LispWorks 4.3, ~~CMUCL 19a~~
- Scheme: GNU guile 1.7.1, MIT Scheme 7.7.1, BBN Scheme, MzScheme 205
- Other Lisp: XEmacs 21.4, GNU Emacs 21.1
- Java 1.5
- Smalltalk: GNU Smalltalk 2.1.10

Implementations of Level 3

- Common Lisp: GNU clisp 2.33.80, LispWorks 4.3
- Other Lisp: XEmacs 21.4, GNU Emacs 21.1

Implementations of Level 4

- Common Lisp: GNU clisp 2.33.80, ~~LispWorks 4.3~~
- Other Lisp: ~~XEmacs 21.4~~, ~~GNU Emacs 21.1~~

Phases of GC

- *Mark phase*: Recursively mark all reachable objects, starting from the root set.
- *Sweep phase*: Move the marked objects to their new location, and update all pointers to point to the new locations. Then free unused memory pages.

Phases of GC

- *Mark phase*: Recursively mark all reachable objects, starting from the root set.
- *Weak object phase*.
- *Sweep phase*: Move the marked objects to their new location, and update all pointers to point to the new locations. Then free unused memory pages.

Weak Object Phase 1st Try

- For all weak-pointers:
  - If the target object is unmarked, break the weak pointer.

Implements level 1 and 2.

Weak Object Phase
2nd Try

- For all weak :key mappings:
  - If the key is marked, mark the value recursively.
- Repeat until stable.
- For all weak-pointers:

  - If the target object is unmarked, break the weak pointer.

  For all weak :key mappings:

  - If the key is unmarked, break the mapping.

Implements level 3. But: $O(W^2)$

Weak Object Phase
3rd Try

- Precompute the reverse mapping from weakly pointed object to weak pointer, as a hash-table for O(1) access.
- Enqueue all marked weak :key mappings.
- Process the queue:
  - If the key is marked, mark the value recursively. While doing that, look up the reverse mappings. Add the discovered weak objects to the queue.

Weak Object Phase
3rd Try (2)

- For all weak-pointers:
  - If the target object is unmarked, break the weak pointer.

  For all weak :key mappings:
  - If the key is unmarked, break the mapping.

Implements level 4: O(W)