A generic heap interface specification for Common Lisp

Ingvar Mattsson
<ingvar@google.com>

December 3, 2012

Chapter 1

Generic extendable heaps for Common Lisp

1.1 Rationale

There is no standard heap implementation in the Common Lisp standard. It is, however, a useful data structure. Having a generic heap implementation that is tweakable to allow for the exact behaviour wanted and peforms relatively well is wanted.

The intention is to provide a portable, flexible, performant heap framework that can be used on essentially all data where storing according to a ranking criterion makes sense.

This API specification carefully does not discuss how it behaves in a multi-processing environment.

1.2 Guarantees

1.2.1 Time complexity

The heap data structure gives you $O(1)$ peek at one extreme of the heap. It also gives you $O(logn)$ addition and removal from the heap.

However, the $O(logn)$ insertion and removal relies on an O(1) comparison operator. With having user-specified comparison (and key extraction) operators, the best guarantee the reference implemenation can give is that insertion and removal is $O(Clogn)$ for a comparator complexity of $O(C)$.

1.2.2 Multi-processing

There are no explicit multi-processing or concurrency guarantees for the generic heaps. However, implementors are encouraged to add recursive locks to each heap object and lock/unlock these as necessary.

1.2.3 Side-effects

Any code that modifies an object currently in the heap is likely to breach the heap invariant. Doing that is highly discouraged. However, modifying things within an object that does not, in any way, contribute to the value used in comparisons may be safe.

1.3 Interface specification

1.3.1 Package

All symbols of a generic heap implementation should be accessible in a package names such that :genheap is a designator for the package.

This means that only one GENHEAP implementation can be loaded in any given image, without having to do package renaming. However, it makes it easier for devbelopers to write code against the package or probe for the package's existence using FIND-PACKAGE.

The functional interface presented is a minimum interface, it may be that an implementor is willing to give guarantees for the implementation that extend beyond this.

The exact underlying data structures are not specified. The reference implementations use CLOS, generic functions and methods on these for its implementation. One of them defines new methods as new generic heap types are asked for, the other one doesn't.

1.3.2 Functional interface

make-heap

(make-heap &key (test #'¡) (initial-size 16) (key #'identity)) -¿ heap

Returns a newly created heap, using the specified test as the heap criterion, using *key* as the value to be compared. The key defaults to the identity function.

insert

(insert value heap) -¿ value

Inserts a new element into the heap. The value inserted is returned.

remove

(remove heap) -¿ (or value nil)

Removes and returns the value at the head of the heap, unless the heap is empty. If the heap is empty, NIL is returned.

emptyp

(emptyp heap) -¿ boolean

This function returns T when called on an empty heap, NIL otherwise.

heapify

(heapify sequence &key (test #'¡) (key #'identity)) -¿ heap

Creates a heap populated by the sequence passed in, returning a heap that should behave identically as one created by MAKE-HEAP, then populated element by element.

peek

(peek heap) -¿ value

Returns the value at the top of the heap, without modifying the heap.

heap-p

(heap-p object) -¿ boolean

This function returns NIL when called on a non-heap object and a true value if presented with a heap object.

(

MACRO declare-heap) (declare-heap &key (test #'¡) (key #'identity))

This macro either updates internal data structures about a specific type of test/key pair or not. It should be called before any heap with that test/key pair is created.

The presence of this macro is to make heap-creation time more predictable, allowing the implementation to do any heavy class-creation up front, instead of at the time of the first heap being created.