1. Hash tables for hash consing.


Note: a different, more elaborated hash-consing library can be found in Why3 sources at http://why3.lri.fr/

Hash consed values are of the following type \texttt{hash\_consed}. The field \texttt{tag} contains a unique integer (for values hash consed with the same table). The field \texttt{hkey} contains the hash key of the value (without modulo) for possible use in other hash tables (and internally when hash consing tables are resized). The field \texttt{node} contains the value itself.

Hash consing tables are using weak pointers, so that values that are no more referenced from anywhere else can be erased by the GC.

\begin{verbatim}
val create : int -> \alpha t
(val clear : \alpha t -> unit
(val hashcons : \alpha t -> \alpha -> \alpha hash\_consed
(val iter : (\alpha hash\_consed -> unit) -> \alpha t -> unit
(val stats : \alpha t -> int \times int \times int \times int \times int \times int

2. Generic part, using ocaml generic equality and hash function.

\begin{verbatim}
type +\alpha hash\_consed = private {
  hkey : int;
  tag : int;
  node : \alpha }

val create : int -> \alpha t
(val clear : \alpha t -> unit
(val hashcons : \alpha t -> \alpha -> \alpha hash\_consed
(val iter : (\alpha hash\_consed -> unit) -> \alpha t -> unit
(val stats : \alpha t -> int \times int \times int \times int \times int \times int

3. Functorial interface.

\end{verbatim}
module type HashedType =
  sig
    type t
    val equal : t → t → bool
    val hash : t → int
  end

module type S =
  sig
    type key
    type t
    val create : int → t
    val clear : t → unit
    val hashcons : t → key → key hash_consed
    val iter : (key hash_consed → unit) → t → unit
    val stats : t → int × int × int × int × int × int
  end

module Make(H : HashedType) : (S with type key = H.t)