https://laurentbloch.org/BlogLB/Vecteurs-de-Bigloo

Vecteurs de Bigloo

- Zinformatiques - Cours de bioinformatique au CNAM -

Date de mise en ligne : vendredi 4 février 2005

Cet article signale quelques extensions au type vecteur du *Scheme* standard qui peuvent être utiles pour réaliser des programmes efficaces.

La plupart des *Scheme* modernes proposent des vecteurs homogènes, c'est-à-dire dont tous les éléments sont de même type, ce qui autorise de meilleures performances, notamment s'il s'agit d'éléments de types scalaires (nombres ou caractères). On pourra consulter à ce sujet le document de normalisation relatif aux vecteurs homogènes de types numériques : Bigloo ne comporte pas cette extension, ce qui ne veut pas dire qu'elle est inutilisable avec lui (il suffit de recopier les programmes sources, éventuellement de les adapter). Incidemment, il y a une adaptation à Bigloo d'une bibliothèque complémentaires d'opérations sur les vecteurs, ainsi qu'un type vecteur homogène numérique non documenté `tvector`, au sujet duquel je cite Manuel Serrano :

Bigloo already contains (undocumented) homogeneous vector types. Even if this feature is currently left undocumented, homogeneous vectors are likely to stick in Bigloo for a very long time because they are at the heart of the CFA optimization that is able to automatically transform Bigloo vectors in homogeneous vectors.

Since there is no documentation, you will have to figure out by yourself how to use typed vectors. To get you started, here is a small example :

```
 (module foo
   (type (tvector vint (int)))
   (main main))

(define (f v)
   (let loop ((i (-fx (vint-length v) 1))
              (a 0))
      (if (=fx i -1)
          a
          (loop (-fx i 1) (+fx a (vint-ref v i))))))

(define (main a)
   (print (make-vint 4 2))
   (print (f (make-vint 10 1))))
```

Sur la liste de diffusion de Bigloo, Will M. Farr a posté cet autre exemple :

â€" So, my question is, does Bigloo have something like an SRFI-4 implementation ?

â€" No, but you can hack one pretty quickly. Use the declaration (for example) :

```
 (type (tvector f64vector (double)))
```

This generates all the standard vector functions for f64vectors. They will not automatically print at the REPL, but they will print using an explicit `(print ...)` statement. The procedure `(make-f64vector n init)` *requires* the

init value (slightly different from SRFI-4). The tvector is stored in a `struct` object which looks like :

```
 struct bgl_tvector_of_BgL_doublez00 {
   header_t header;
   long     length;
   obj_t    descr;
   double el0;
};
```

Just use a pragma form (described in the C interface) to get a pointer to `el0`, and you're ready to pass them to C functions.