comp.lang.scheme

Re: inline cache for method lookups

Pinku Surana <sura...@gmail.com>

Sunnan wrote:
> Pinku Surana wrote:
> > Is there a Scheme 00 package that supports caching of method lookups at
> > a call site, similar to monomorphic inline caches in Smalltalk?

> > I'd like to write something like this:

> > (define (sum . lst)
> > (with-inline-cache +
> > (reduce + 0 lst)))

> > Since lst is likely to contain the same types, the cache can store the
> > last successful method and type test for this call site. The next time
> > + is called, it applies the cached type test. If true, apply the cached
> > method; otherwise, do a normal generic call.

> > Thanks,
> > Pinku

> I'm possibly misunderstanding what you want to do, but most OO-packages
> should work if you can memoize methods.

The inline cache used in Smalltalk and Self is *not* the same as memoization. If I memoize the + generic method, then it will map the inputs to outputs: (+ 1 2) --> 3. If I call (+ 1 2) again, it returns the answer found in the cache; otherwise, it calls the generic +.

Every time I call a generic method like +, it has to do some big lookup to figure out which method to use. 20 years ago Smalltalk added a cache at the call site to remember which method was selected the last time. If I call (+ 1 2), then the cache will store a pointer to the function like this:

```
(lambda (x y)
  (if (and (integer? x) (integer? y))
    (int+ x y)
    (call-with-values (lambda () (+ x y))
      (lambda (quick-dispatch answer)
        (reset-cache-with-new-dispatch er quick-dispatch)
        answer))))
```

If I call + at that call site with integers again, it will skip the

expensive generic call and jump straight to int+. If I call + with reals, it will call the generic + and reset the cache to check for (and (real? x) (real? y)).

In Self, they expanded this to polymorphic caches. In my example, when I call + with two real numbers it will expand the cached dispatcher to look like this:

```
(lambda (x y)
  (cond
    ((and (integer? x) (integer? y)) (int+ x y))
    ((and (real? x) (real? y)) (real+ x y))
    (else (+ x y))))
```

I hacked the first (monomorphic) version of this into Scheme48's generic methods quite easily. If I can do it, I assume it's already been done elsewhere ;-). Both the Smalltalk and Self papers on the topic argue that the caches are effective and improve performance substantially. If it helps these latently typed languages, it should help Scheme OO packages, too.

Anyway, this is what I'm looking for. Not memoization.

Pinku