

expand-only

November 16, 2006

Scheme's built-in *expand* takes an expression containing calls to macros and performs the macro-expansion step on that expression. Unfortunately, *expand* expands *every* macro call, and there is no way to expand only a select few macros. The file `expand-only.ss` contains an alternative macro expander, called *expand-only*, that allows us this capability. We can use *expand-only* to expand some examples of miniKanren code to examine its implementation. The function *expand-only* takes a quoted list of macro names to expand and a quoted Scheme expression. It expands only the macros listed in the list that it is passed, and returns this expanded expression.

cond^e

First, we will examine a simple **cond**^e expression:

```
(conde
  [g1]
  [g2])
```

First, we will expand only **cond**^e:

```
> (expand-only '(conde) '(conde [g1] [g2]))
⇒
(λG (p)
  (inc (mplus* (bind* (g1 p)) (bind* (g2 p))))))
```

Next, we expand *mplus**:

```
> (expand-only '(conde mplus*) '(conde [g1] [g2]))
⇒
(λG (p)
  (inc (mplus (bind* (g1 p)) (λF ()
    (bind* (g2 p)))))))
```

Finally, we expand *bind**:

```
> (expand-only '(conde mplus* bind*) '(conde [g1] [g2]))
⇒
(λG (p)
  (inc (mplus (g1 p) (λF ()
    (g2 p))))))
```

Notice that both goals *g1* and *g2* are applied to the same package (or substitution) *p*. The infinite streams of substitutions returned by these applications are independent of one another, just like the lines of a **cond**^e.

fresh

Next, we expand a simple **fresh** example:

```
(fresh (x)
  g1
  g2
  g3)
```

First, we expand **fresh**:

```
> (expand-only '(fresh) '(fresh (x) g1 g2 g3))
⇒
(λG (p)
  (let ((x (var (quote x))))
    (bind* (g1 p) g2 g3)))
```

Next, we expand *bind**:

```
> (expand-only '(fresh bind*) '(fresh (x) g1 g2 g3))
⇒
(λG (p)
  (let ((x (var (quote x))))
    (let ((a-inf (g1 p))
          (and a-inf (let ((a-inf (bind a-inf g2)))
                      (and a-inf (bind a-inf g3)))))))
```

Notice that the infinite stream of substitutions *a-inf* is updated with each call to *bind*. This means that the stream of substitutions fed to *g2* is affected by the result of *g1*, and the input for *g3* is affected by the result of *g2*. The goals inside of a **fresh**, then, will affect one another, just as they should.