Patrisika : 如何实现 CPS 变换

Belleve Invis · 2013-08-20

Moescript 里的 CPS 变换是基于 CFG 的。在处理的时候，所有原本的结构都会灰飞烟灭，只留下一堆残迹。这样做优点是可以处理任意复杂的控制结构（甚至可以兼容 GOTO），缺点也很明显，原本程序里的控制流被毁的一干二净，让调试非常不爽；而且这种处理会产生成堆的临时变量（Patrisika 里对应 `.t` 算符），这也会让最终生成的程序非常难看。

因为 Patrisika 使用了正交且类似 S-exp 的 AST 格式，许多 Moescript 里必须重复编码的流程控制在 Patrisika 里得以称为可能。不过在说控制流前，我们要先处理最简单的节点。Patrisika 的节点大多是这样：

```
[operator, arg1, arg2, arg3, ..., argn]
```

按照约定，各个参数的求值顺序永远是从左到右，等他们算完后一块丢进算符 operator，再扔给某个 Continuation。所谓 Continuation就是一个单参函数，写过回调的都知道是什么样子。（什么，nodejs 的回调是两个参数的？先不考虑异常这种煞风景的东西。）在 Patrisika 里，函数的算符是 `.fn`：

```
['.fn', ['.list', tCont], b]
```

（你可能会纳闷那个 `.list` 是怎么回事。那个东西是为了实现在参数表上进行模式匹配而制造的。）

对某个语法树节点进行 CPS 变换需要两个参数：节点，以及等待其值的上下文（Context）。Context 即 Continuation，CPS 变换就是利用某种方法让他们联姻。对于最简单的符号节点，CPS 变换毫无难度：

```
cps("symbol", ctx) = [ctx, "symbol"]
```

下面考虑节点 `[operator, a]`。它形式足够简单，只有一个算符和一个参数。`[operator, a]` 的值由算符 `operator` 和子节点 `a` 决定，因为算符是固化在编译器里的，所以它不会被 CPS 变换处理。因此，这个节点的计算顺序是：

- 计算子节点 `a` 的值，存入 `tA` 中
- 将 `tA` 丢进算符 `operator` 得到返回值
- 把返回值丢给上下文

这上面有三次传递。第三次可以写成：

```
[ctx, <return_value>]
```

`return_value` 的定义是：

```
[operator, tA]
```

因此，第二步和第三步的总结果是：

```
[ctx, [operator, tA]]
```

整个 CPS 变换里最精妙的是第一步和第二步之间的连接。第一步——「计算子节点 `a` 的值，存入 `tA` 中」，和我们这里的总任务，「计算 `[operator, a]` 并传入 `ctx`」形式一致，故可以递归完成。而「存入 `tA`」则可用另一个 Continuation 表达：

```
['.fn', ['.list', tA], [ctx, [operator, tA]]]
```

因此我们得到：

```
cps([operator, a], ctx) = cps(a, ['.fn', ['.list',
tA], [ctx, [operator, tA]]])
```

于是，我们把变换 `[operator, a]` 的任务转化为了变换 `a`，递归地做下去就行了。

对于有多个参数的节点，例如 `[operator, a, b]`，它的 CPS 变换过程是相似的。只不过这里要有两个临时变量，来存储两个子节点 `a` 和 `b` 的返回值。它的 CPS 变换结果为：

```
cps([operator, a, b], ctx) = cps(a, ['.fn', ['.list',

tA],

    cps(b, ['.fn', ['.list', tB],

        [ctx, [operator, tA, tB]]])])
```

在 Patrisika 里，「标准」节点的 CPS 变换范式是：

```
// 函数 mt()：获取一个临时变量，[.t] 节点
//   assignCont(node,   continuation)  :   等  价  于

[continuation, node]，包含 IIFE 展开功能

var cpsStandard = function(node, continuation){

    var c = assignCont(node, continuation);

    for(var j = node.length - 1; j >= 1; j--) {

        var t = mt();

        c = cps(node[j], Continuation(t, c))

        node[j] = t;

    }

    return c;

}
```

不过，除了「标准」节点，Patrisika 里还有成堆的节点需要特殊的处理，比如 If。Patrisika 的 If 节点范式是：

```
['.if', condition, thenPart, elsePart]
```

它的 CPS 展开就相对特殊了。按照标准的思路，展开它的结果应该是这样：

```
cps(condition, ['.fn', ['.list', tC],
   ['.if', tC,
      cps(thenPart, ctx),
      cps(elsePart, ctx)]])
```

然而这里要对 `ctx` 进行深拷贝，代码膨胀的问题会异常严重，而且复制节点也会导致 Source Map 失效。因此最后的解决方案是：

```
[['.fn', ['.list', tCtx],
   cps(condition, ['.fn', ['.list', tC],
      ['.if', tC,
         cps(thenPart, tCtx),
         cps(elsePart, tCtx)]])
], ctx]
```

对于 `.while` 节点则需要使用递归来完成 CPS 变换。具体结果，各位来想出来好了。