预告：从解释器到抽象解释

Belleve Invis · 2014-05-25

```
function interpret(form, env, ctx){
    if(form instanceof Array){
        switch(form[0]){
            case 'lambda': {
                var params = form[1];
                var body = form[2];
                return     ctx(function(ctx){     return
                function() {
                    var e = Object.create(env);
                    for(var  j  =  0;  j  <  params.length;
                    j++)
                        e[params[j]] = arguments[j];
                    return interpret(body, e, ctx)
                }})
            };
            case 'if': {
                var test = form[1];
                var consequent = form[2];
                var alternate = form[3];
                return      interpret(form[1],      env,
                function(c){
                    if(c)  return  interpret(consequent,
                    env, ctx)
                    else    return  interpret(alternate,
                    env, ctx)
                })
            };
            case 'callcc': {
                return      interpret(form[1],      env,
                function(f){
                    var fctx = function(){
```

```
                    return function(x){ return ctx(x)
                    }};
                return f(fctx)(fctx)
            })
        };
        case 'quote': {
            return ctx(form[1])
        };
        default: {
            return interpretCall(form, env, ctx);
        };
    }
} else if(typeof form === 'string') {
    return ctx(env[form])
} else {
    return ctx(form)
}
}

function interpretCall(form, env, ctx){
    return interpret(form[0], env, function(callee){
        return      interpret$(form.slice(1),      env,
        function(args){
            return callee(ctx).apply(null, args)
        })
    })
}
function interpret$(form, env, ctx){
    if(!form.length) return ctx(null);
    else return interpret(form[0], env, function(x0){
        return      interpret$(form.slice(1),      env,
        function(x$){
            if(x$) return ctx([x0])
            else return ctx([x0].concat(x$))
        })
    })
}

function id(x){ return x }

var env0 = {
    trace: function(ctx){
```

```
        return        function(x)        {        return
        ctx(console.log(x)) }},
    begin: function(ctx){
        return        function(x$)        {        return
        ctx(arguments[arguments.length - 1]) }}
};

interpret(['trace', ['callcc', ['lambda', ['return'],
['begin',
    ['trace', ['quote', 1]],  // traces
    ['return', ['quote', 2]], // returns, sent to
    'trace'
    ['trace', ['quote', 3]]   // not executed
]]]], env0, id);
```

上面的这一堆东西，已经足够做一个完整的解释器了：我们实现了词法作用域、Lambda 抽象以及 `call/cc`。有趣的是，这个解释器的框架代码可以沿用到编译器里：从本质上来说，「编译」也是「解释」的一种。

嘛，在这里先卖个关子，各位记得来 HangJS 来听我详细解说哦～