Login

Assignment 9: ParentheC Interpreter

> “Code should run as fast as necessary, but no faster; something important is always traded away to increase speed.”

Introduction

Preliminaries

This assignment relies on your successful completion of a7. If you haven't successfully completed a7 and maintained the versions of your code along the way, please complete this before starting this assignment. **You cannot receive a grade for this course until you complete this assignment and demonstrate your understanding with one of the instructors. See the end of this page for more details.**

- If you haven't done so, you might consider reading the ParentheC paper, *Using ParentheC to Transform Scheme Programs to C or How to Write Interesting Recursive Programs in a Spartan Host* . It is slightly out of date viz. registerization, but can still prove a useful resource.
- Download pc2c.rkt, and parenthec.rkt.
- You will also need to use the following `define-union` for expressions and `main` program:

```
(define-union expr
  (const cexp)
  (var n)
  (if test conseq alt)
  (mult nexp1 nexp2)
  (sub1 nexp)
  (zero nexp)
  (letcc body)
  (throw kexp vexp)
  (let exp body)
  (lambda body)
  (app rator rand))
```

```

;; (let ((f (lambda (f)
;;            (lambda (n)
;;              (if (zero? n)
;;                  1
;;                  (* n ((f f) (sub1 n))))))))
;;   (* (letcc k ((f f) (throw k ((f f) 4)))) 5))

(define main
  (lambda ()
    (value-of-cps
     (expr_let
```

```
      (expr_lambda
       (expr_lambda
        (expr_if
         (expr_zero (expr_var 0))
         (expr_const 1)
         (expr_mult (expr_var 0) (expr_app (expr_app (expr_var 1) (expr_var 1)) (expr_sub1 (expr_var 0)))))))
      (expr_mult
       (expr_letcc
        (expr_app
         (expr_app (expr_var 1) (expr_var 1))
         (expr_throw (expr_var 0) (expr_app (expr_app (expr_var 1) (expr_var 1)) (expr_const 4)))))
       (expr_const 5)))
     (empty-env)
     (empty-k))))
```

Notice that this test program is not quoted data.

Assignment

Your assignment is to complete the transformation of your interpreter from a7 to a version we can translate to C. When your interpreter is complete, turn it into C programs using `pc2c`, and run the test program provided. Here are the steps you will need to accomplish. Save a new copy of your interpreter after you finish every step. **We will expect you to have all of these intermediate files available during your demonstration**. Also, you will often need to go back to an older version of your interpreter and having copies of all of them will save a lot of time.

1. Below `#lang racket`, add the line

   ```
   (require "parenthec.rkt")
   ```

   Next add the `expr` `define-union` to your file, change the `match`-expression in `value-of-cps` to instead be a `union-case`-expression. Consult the ParentheC paper or the example from class to see how to do this. Make sure to remove the backquotes and commas in the patterns of what was your `match` expression. Add `main` to the bottom of your file, and make sure it returns `120` when you invoke it.

2. Transform your closure constructor to a `define-union`, change the `match` in `apply-closure` to instead use `union-case`, and ensure that your constructor invocations are preceeded with `clos_`, or something other than `clos` if you use a different name for your union. Make sure to remove the backquotes and commas in the patterns in what was your `match` expression.
3. Transform your environment constructors to a `define-union`, change the `match` in `apply-env` to instead use `union-case`, and ensure all constructor invocations are preceeded with `envr_`, or something other than `envr` if you use a different name for your union. Make sure to remove the backquotes and commas in the patterns in what was your `match` expression.
4. Transform your continuation constructors to a `define-union`, change the `match` in `apply-k` to instead use `union-case`, and ensure all constructor invocations are preceeded with `kt_`, or something other than `kt` if you use a different name for your union. Make sure to remove the backquotes and commas in the patterns in what was your `match` expression.

5. Transform all your serious function calls to our A-normal form style, by adding `let*` above your serious calls, and ensuring that the names of the actual parameters to the serious calls are *exactly* the names of the formal parameters in the definition.
6. Registerize the interpreter. Turn each `let*` expression to a `begin` block: the former `let*` bindings will become `set!` expressions, and the body becomes the invocation of a function of no arguments. Change all serious functions to be functions of no arguments. Define your global registers using `define-registers` at the top of the program.
7. Change all of your `(define name (lambda () …))` statements to instead use `define-label`. Define your program counter at the top of the program using `define-program-counter`.
8. Convert all label invocations into assignments to the program counter, and then add calls to `mount-trampoline` and `dismount-trampoline`. Note this will require modifying `empty-k` in your `kt` union, and the `empty-k` clause in the `union-case` inside `apply-k`. On the last line of `main`, print the register containing the final value of the program, e.g. `(printf "Fact 5: ~s\n" v)` See the parentheC document for notes on these steps.
9. Comment out the lines `#lang racket`,

   ```
   (require "parentheC.rkt")
   ```

   and your invocation of `main` (that is `(main)`) if you added it to your file. And save a copy of this file named `interp.pc`.

10. Open and run `pc2c.rkt`. This should load without errors. In the associated Racket REPL with **no** other files loaded, type

    ```
    (pc2c "interp.pc" "a9.c" "a9.h")
    ```

    which will generate C code from your interpreter. Compile the C program with the C compiler of your choice. The SOIC linux machines have `gcc` installed, and you can find here [http://gcc.gnu.org/install/binaries.html] binaries for many different systems. Alternately, you could use an online C compiler [http://tutorialspoint.com/compile_c_online.php] Run the resulting executable, verifying that you see the correct output.

Save a new copy of your interpreter after you finish each step. **We will expect you to have all of these intermediate files available during your demonstration**. Also, you will often need to go back to an older version of your interpreter and having copies of all of them will save a lot of time.

- You should turn in a file named `interp.pc` that contains the exact Scheme code you used to generate your C programs.
- This assignment is due on **Wednesday, 3/20 at 11:59pm** Once you've done the assignment, you must meet with one of the AIs to demonstrate your knowledge of your code. This meeting is a required part of the assignment and must take place on or before **4/10**.

- Remember: successful completion of this assignment and code review is required in order to receive a grade for this course. For assignments handed in after the due date, credit is reduced proportionally to lateness (1 assignment grade/week).

Just Dessert

Add a `callcc` form to your interpreter that behaves like Scheme's `call/cc`. Change the test program to one that uses `callcc` and send this, along with any other required changes, in an email to your grader.

assignment-9.txt · Last modified: 2019/02/27 19:12 by mvc