CS 410/510 Compiling Advanced Languages Homework 4 – due Tuesday, May 30, 2000

Pencil and Paper

1. Consider the closure conversion technique described in class (the one which produces simple flat closure records). Show how the closure records can be assigned existential types so that the closure-converted version of the expression

`λa:Int. ((λf:(Int->Int). f 3) (if b then λx:Int.x+2 else λx:Int.x+a))`

is well-typed.

Programming

The files on the web page give a translator from the original core language to PNF, and an interpreter for the latter. To execute an example, such as `d`, you load all the files and then type something like:

```
Pnfinterp.evaluate (Tcore2pnf.translate a);;
```

which should produce the same answer as HW 3's `Interp.evaluate a`.

2. Extend `Pnfinterp` to work on mutually recursive functions, such as the one defined in example `d1`.

3. The closure records built in `Pnfinterp` contain a reference to the complete environment. In fact, it would be adequate to store only a "thinned" environment containing entries for just the free variables of the function. Write routines to calculate the free variables of an arbitrary PNF expression and trivial expression, and use them to produce such a "thinned" environment and store it in the closure. Verify that the interpreter still works.

Extra Credit: The type `IntT` assigned to all newly introduced names in `tcore2pnf` is clearly bogus; in general, PNF code won't typecheck. (Of course, I haven't provided a typechecker anyhow!) Fix this and verify your fix. (Hint: The best way is probably to change the representation of `Var` nodes in `Tcore` so that they carry their own type; this would permit the type of an arbitrary expression to be computed in a context-independent way. An alternative way, maybe faster to implement, is for the translation routine to carry around a typing environment.)