HW 7
50pts. You can work on your own or in teams of two.
Posted Tuesday, November 29, 2022
Due Friday, December 9, 2022

Problem 1 (7pts). Consider the twice combinator:
\[ \text{twice} = \lambda f. \lambda x. f(f\ x) \]
Reduce expression \(\text{twice twice f x}\) into normal form using normal order reduction. For full credit, show each step on a separate line.

Problem 2 (8pts). Now consider the Haskell implementation of twice:
\[ \text{twice f x} = f(f\ x) \]
(a) What is the type of twice?
(b) What is the type of expression twice twice?
(c) If the type of fun is Int->Int, what is the type of expression twice twice fun?
(d) If the type of fun is Int->Int and expression twice twice fun v is well-typed, what is the type of twice twice fun v?

Note: You do not need to justify your answer, just state the corresponding type expression.

Problem 3 (10pts). This is a skeleton of the quicksort algorithm in Haskell:
\[
\text{quicksort \ [\] = \ [\]}
\text{quicksort (a:b) = quicksort \ldots \ ++ [a] ++ quicksort \ldots}
\]
(a) Fill in the two elided expressions (shown as \ldots) with appropriate list comprehensions.
(b) Now fill in the two elided expressions with the corresponding monadic-bind expressions.

Problem 4 (5pts). In the following code, which of the variables will a compiler consider to have compatible types under structural equivalence? Under strict name equivalence? Under loose name equivalence?

\[
\text{type A = array [1..10] of integer}
\text{type B = A}
\text{a : A}
\text{b : A}
\text{c : B}
\text{d : array [1..10] of integer}
\]

Problem 5 (10pts). Show the type trees for the following C declarations:
\[
\text{double *a[n];}
\text{double (*a)[n];}
\text{double (*a[n])();}
\text{double (*a[])[n];}
\text{double (*a(int, double(*)(double, double[])))(double);}
Problem 6 (10pts). Consider the Pascal-like code for function `compute`. Assume that the programming language allows a mixture of parameter passing mechanisms as shown in the definition.

```plaintext
double compute(first : integer /*by value*/, last : integer /*by value*/,
incr : integer /*by value*/, i : integer /*by name*/, term : double /*by name*/)

result : double := 0.0
i := first
while i <= last do
    result := result + term
    i := i + incr
endwhile
return result
```

(a) (2pts) What is returned by call `compute(1, 10, 1, i, A[i])`?
(b) (2pts) What is returned by call `compute(1, 5, 2, j, 1/A[j])`?
(c) (2pts) `compute` is a classic example of Jensen’s device, a technique that exploits call by name and side effects. In one sentence, explain what is the benefit of Jensen’s device.
(d) (4pts) Write `max`, which uses Jensen’s device to compute the maximum value in a set of values based off of an array `A`. 