CS314 Spring 2014

Assignment 9

WILL NOT BE GRADED

Sample solution will be posted by Monday, May 5

Problem 1 – Vectorization

A statement-level dependence graph represents the dependences between statements in a loop nest. Nodes represent single statements, and edges dependences between statements. An edge is generated by a pair of array references that have a dependence. Edges are directed from the source of the dependence to its sink. For example, for a true dependence, the source is a write reference, and the sink is a read reference. There may be multiple edges (i.e., dependences) between two nodes in the graph.

```
        for i = 2, 99
S1:        a(i) = b(i-1) + c(i+1);
S2:        b(i) = c(i) + 3;
S3:        c(i) = c(i-1) + a(i);
        endfor;
```

Here is a basic vectorization algorithm based on a statement-level dependence graph:

1. Construct statement-level dependence graph considering true, anti, and output dependences; in the final dependence graph, the type of the dependence is not important any more
2. Detect strongly connected components (SCC) over the dependence graph; represent SCC as summary nodes; walk resulting graph in topological order; For each visited node do
   (a) If SCC has more than one statement in it, distribute loop with statements of SCC as its body, and keep the code sequential.
   (b) If SCC is a single statement and has no dependence cycle, distribute loop around it and "collapse" loop into a vector instruction. For example, the loop

   ```
   for i=1, 100
     a(i) = b(i) + 1;
   endfor
   ```

   can be "collapsed" into a single vector instruction

```
a(1:100) = b(1:100) + 1;
```

. If there is a dependence cycle on the single statement, distribute the loop around the statement and keep loop sequential.

1. Show the statement-level dependence graph for the loop with its strongly connected components.
2. Show the generated code by the vectorization algrithm described above.

Problem 2 – Type Systems

Assume that E is a type environment that maps variables and constants to their type expressions. Assume that the environment already contains the following mappings

E = { (1 → integer), (2 → integer), (true → boolean), (false → boolean) }

---

1. integer addition:
$$\frac{E \vdash e_1 : integer \quad E \vdash e_2 : integer}{E \vdash (+\ e_1\ e_2) : integer}$$
2. Polymorphic cons:
$$\frac{E \vdash e_1 : \alpha \quad E \vdash e_2 : list(\alpha)}{E \vdash (cons\ e_1\ e_2) : list(\alpha)}$$
3. Polymorphic car:
$$\frac{E \vdash e : list(\alpha)}{E \vdash (car\ e) : \alpha}$$
4. Polymorphic '():
$$E \vdash\ '() : list(\alpha)$$

---

1. Give type rules for polymorphic function cdr and polymorphic function null? functions.
2. Apply the above type inference rules to determine whether the following programs can be typed. List every step explicitly. In case of a type error, show the situation where no rule can be applied any more, i.e., where your type inference process get's stuck.
   (a) (car (cons (+ 1 2) '()))

(b) (cons true (cons 1 '()))

(c) (cdr (cons 1 (cons 2 '())))

(d) (cons 1 2)

(e) (cons true '())

(f) (null? '(1))